1970

A comparison of time-sharing and batch processing through an analysis of programming errors

Donald M. Jackson
Lehigh University

A COMPARISON OF TIME-SHARING AND BATCH PROCESSING THROUGH AN ANALYSIS OF PROGRAMMING ERRORS

by

Donald M. Jackson

A THESIS

Presented to the Graduate Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Industrial Engineering

Lehigh University
1970

A COMPARISON OF TIME-SHARING AND BATCH PROCESSING THROUGH AN ANALYSIS OF PROGRAMMING ERRORS

by

Donald M. Jackson

A THESIS

Presented to the Graduate Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Industrial Engineering

Lehigh University
1970

This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science

~~June 1970~~

May 2, 1970

William A. Smith Jr.
Professor In Charge

Chairman of the Department

Acknowledgements and Credits

To Prof. William A. Smith, I wish to gratefully acknowledge the guidance and cooperation extended to me. It was only through his encouragement that I was able to research, design, and conduct an experiment which perhaps added a little knowledge to man's insight into human problem solving with the computer.

I would also like to extend my gratitude to Western Electric for its use of the Education Center facilities and particularly the eight gentlemen who served as experimental subjects.

I am indebted to the comments and criticisms expressed by Dr. Harold Sackman at Systems Development Corporation. It is his research and writings that I have called heavily upon in my experimentation.

Last to my wife, who typed and proofread the manuscript through its several stages, I owe special thanks for her patience and good humor.

June 1970

Table of Contents

Appendix Listing

List of Tables and Figures

List of Tables and Figures

Abstract

This thesis describes an experiment conducted to compare the response and performance of a programmer under the two current modes of program development: conversational time-sharing and batch processing. This comparison was made through a measurement and analysis of programming errors. The research reported herein had two specific objectives. First to develop a procedure for identifying, classifying, and analyzing programming errors as a measure of performance, and secondly to ascertain experimentally if certain types of programming errors are more frequently occurring and more difficult to correct under conversational time-sharing than under the batch mode of processing.

From an analysis of the growing body of scientific knowledge in this area and observation of student programmers, an error classification scheme was developed and tested. Then a series of experiments was designed and conducted at the Western Electric Graduate Education Center. While the experimental analysis revealed some insight into the phenomena of error detection and correction, few conclusive results were obtained due to insufficient data.

A TRANSFORMATION error is created from an incorrectly translated set of program steps even though the programmer correctly perceived the operations necessary to perform the desired function. This kind of error took more runs or executions of the program to detect and correct under time-sharing than under batch operation. The conclusion is that it takes longer (more runs) to find these errors because the subject is so "close" to the segment of the program in front of him (not the whole program listing). He has the very convenient and quick option to re-execute his program rather than look over it all for errors (desk checking).

Also identified are areas where further research is urgently needed and guidelines for proper experimental design in study of the man/computer systems.

Section 1

Objectives

It was the intent of the research and experimentation reported herein to expand upon the growing body of knowledge related to the interaction of man and machine in the computing environment. These experiments deal specifically with a comparison of programmer response and performance under the two programming modes - conversational time-sharing and batch processing. This comparison is facilitated through an analysis of programming errors and how they influence programmer response and performance.

The research had two objectives:

1. To develop a suitable method and procedure for using program error as an experimental measurement and an indicator of performance. This required design of a classification scheme which would permit one to logically segregate and classify types of errors for the purpose of determining user characteristics.
2. To investigate experimentally the existence of a phenomenon entitled by the author,

"The Principle of Forced Interaction"
which has been defined as:
An overview of program construction, particularly an understanding of the logical relationships of program modules, is substantively restricted by the incremental compilation and instant turnaround characteristics of conversational programming on the time-sharing system.

Section 2

Background and Critique of Subject Research

With the advent of asynchronous input/output operations on computing systems in the late fifties, it became possible to perform simultaneously the operation of a computer's central processor and its input/output processors. Programmers immediately set themselves to the task of using this new hardware capability to maximum advantage. Initially this meant better organization of individual programs. Input/output operations were strategically placed to achieve maximum use of the central processor. It did not take long, however, for the idea to be extended to that of operating two programs "simultaneously" so that one performed input/output operations while the other used the central processor. The objective was to maintain the fullest possible useful activity in those parts of the computing system which could function simultaneously. Under this kind of a set up the central processor was required to transfer control frequently from one program to another without necessarily waiting for any of the programs to terminate. Since portions of programs were sharing the central processor sequentially in time with all other program portions, the term time-sharing arose.

Concurrent to the development of the above techniques, people were experimenting with the idea of connecting many electric typewriters or teletypewriters to a computer and using them as on-line input/output devices. It was recognized at the onset that, if men were going to use these teletypewriters as on-line devices, there would be a lot of very slow input and output operations. Not only would programs be delayed because of the slowness of the transfer rates of these devices, but they would be further delayed by the users for slow rates of typing. Time-sharing of programs "simultaneously" in the computer was the logical technique to employ for this situation.

Time-Sharing

Time-sharing, however, is embedded in an on-line environment, and there are special time constraints imposed on the design. Previously, it was only necessary to switch from program to program to maximize utilization of all the components of the hardware. In the on-line environment there is the additional requirement to pass control to the jobs of the different users at frequent intervals so as not to ignore any user for a substantial length of time lest he become disenchanted

with the system. With this additional demand, even if a single job program were exercising the components of the computer to their greatest extent with maximum efficiency, that program would have to be suspended periodically in order to make the system available to other users.

This additional aspect to the notion of time-sharing is quite significant. It is so significant, in fact, that the requirement to respond to all users of the system has become the major characteristic of time-sharing systems and the term has become identified with this characteristic almost to the exclusion of the previous notion--that of maximizing utilization of all the components of the hardware. The term "time-sharing" can be defined then as:

> The operation of a computer facility that permits many users to operate the facility simultaneously, or apparently simultaneously, in such a way that each is or can be completely unaware of the use of the facility by others.(2)

There were primarily three experimental programs where the time-sharing system was conceived and developed.

1. MIT developed the Compatible Time-Sharing System (CTSS) used for Project MAC (1962)

2. SDC developed Time-Sharing (TSS) for Advanced Research Project Agency of the Department of Defense (1964)
3. RAND Corporation developed JOSS, the Johnniac Open-Shop System (1964) (13)

Commercial applications have emerged and now most major computer hardware manufacturers are offering some version of time-sharing facilities.

In time-sharing utilities, particularly, the term time-sharing also means expense sharing of overall systems costs among a large number of users. Time-sharing facilitates quick and direct access to the computer when the user wants it. For most computing tasks, the user (often remote) can get the results he wants in a matter of minutes rather than hours or days. In the time-sharing environment the user may exert a degree of control over his program and may change his mind or at least do things differently as he interacts with the computer (within the limits of the system).

In time-sharing systems, the expanded hardware and very extensive software are generally more costly to build and operate than batch systems using a central

computer. This is particularly true when comparing smaller systems of either type. Proponents of the time-sharing concept claim that the higher cost is justified in terms of convenience to the user, faster (elapsed time) program development and reduction, therefore, in manpower requirements.

Conversational Programming

Because of the on-line interactive nature of time-sharing, direct man/computer communication through a "pseudo-natural language" format and pace is a reality. This is achieved through conversational programming.

Conversational mode programming means on-line generation of a program with incremental compilation and execution in a time-sharing environment. It involves a high degree of man/machine interaction since program compilation, modification, and execution are all controlled via a set of easily learned commands issued by the user at the remote console.

Conversational mode FORTRAN IV permits incremental compilation of source program statements, which means that one or more program statements can be compiled independently. To modify a compiled statement the user recompiles only that statement. To add test to a program

the user issues a command that directs the system to insert new material at a specified location. The results are very fast turnaround time for the programmer and minimal work for the system in keeping up with the program changes.

Conversational FORTRAN IV also has a convenient set of on-line debugging features. The user is allowed to set breakpoints, insert temporary statements that execute, but do not become compiled as a permanent part of the program, and resume execution at any specified point (also implicit formatting).

Batch Processing

Thus, we can now compare the two modes of computer programming of interest. Batch processing, traditionally, is a mode of operation in which a program is submitted and placed in the queue of programs waiting to be run. When it is finally run, the entire computer system is dedicated (or a specified part in a multi-programming environment) to the computation and/or execution of this job. Under the batch mode every job is run to completion in the sequence submitted to the computer. The batch system is characterized by relatively fast input and output devices and some unique debugging tools such as core dumps and tracing lists.

In batch type systems, jobs are stacked and processed more or less sequentially on a waiting line basis where the user has only indirect contact tiwh the computer. Under batch processing, maximum system economics are realized with a minimum waste of computer resources. However, turnaround times will vary from hours to days or even weeks depending on such things as:

1. operating system requirements
2. job priority
3. estimated computer running time

Under batch systems the useful computations per unit time are maximized by the most efficient use of the computing system.

Time-Sharing Modification

Because a significant portion of the computational capability of a time-sharing system is devoted to relatively non-productive functions such as program swapping, idle time, etc., the cost effectiveness appears to be considerably lower than the fast turnaround batch systems (RJE-remote job entry). Again on the basis of total costs for the system hardware, software, etc., the comparable batch systems appear to operate more cheaply in the computer than its time-sharing counterpart.

The following list characterizes some of the more frequently voiced claims for the time-sharing system:

1. The savings in programmer man-hours and in the time required to produce working programs under time-sharing more than offset increased operating costs over batch mode.
2. Programmers grow lazy and adopt careless and inefficient work habits with the availability of computing capability under time-sharing.
3. Easy access to the computer tends to make users more prone to casual and costly trial and error computer runs with poorly prepared problems in an effort to trade off computer time against human time.
4. Batch systems encourage more extensive desk checking between runs, and thus in effect, errors are found and corrected quicker.

Comparison of Studies

Today, in effect, many computer installations are pursuing some of the features of both approached:

1. On-line time-sharing foreground, batch background.
2. Operating systems incorporating direct coupled computer concepts and remote job entry.

3. High capacity, fast turnaround batch systems.

What about quality experimental comparison work to evaluate the features and drawbacks of the two approaches? In man/computer communication, little experimental evidence exists. There have been five pertinent experimental studies reported and, although collectively they do provide an instructive body of knowledge, a critical examination of each will reveal substantial gaps in these experiments which substantiate the need for the research reported herein. Table 2-1 summarizes the environmental characteristics for each of the experiments and a comparison to the proposed study. The summary of prior work was developed by Sackman for the Spring Joint Conference 1968.(14)

Notice that in most all the experiments the choice of subjects was limited to students or programmer trainees who had a correspondingly brief period of programming experience; hence one might expect greater response to experimentation and learning effect but perhaps less bias to methods previously used. An objective of this research was to perform experiments under the engineering and technical conditions of the industrial setting. This objective would preclude the use of undergraduate programmers or novice trainees. On the other extreme, those requirements stated would also preclude the use of highly

Table 2-1

COMPARISON OF RECENT & CURRENT RESEARCH: Subject and Problems

EXPERIMENTAL CHARACTERISTIC	SCHATZOFF, TSAO, AND WIIG (16)	ERIKSON (12)	GRANT AND SACKMAN (12)	SMITH (18)	GOLD (4)	JACKSON
Sample Size	4	9	12	127	60	8
Type of Subjects	undergrad. students with high programming apt.	programmer trainees	experienced programmers from R&D setting	undergrad. & grad. students in an introductory programming course	undergrad. & grad. students	special grad. students with industrial background (Western Electric Ed. Ctr.)
Experience Level	"some" programming experience	less than one year	average of 7 yrs	most subjects had less than a year experience	78% of subjects had taken at least one programming course	class instruction & at least one year of experience
Number and Types of Problems	4 problems, Monte Carlo integration, algebraic sorting, Pig Latin translator, test format convertor	2 problems - a sorting routine & a cube puzzle	2 problems - algebra & maze	2 easy warm-up problems, 4 experimental problems-cosine infinite series, matrix sorting, language translation, heuristic program	1 problem - simulation model of construction industry	2 problems - invoice preparation with file manipulation, inventory simulation with Monte Carlo
Difficulty Level	moderately difficult for skilled student subjects	conceptually simple	moderately difficult for highly experienced programmers	moderately dif. ficult for beginners	moderately difficult - open-ended problem	moderately easy for experienced programmers
Average Comletion Time	approx. 40 hrs. to complete all problems	a few hours	approx. 60 hrs. to complete both problems	approx. 60 hrs. to complete both problems	15 to 20 hrs.	approx. 10 hrs. to complete both problems

experienced dedicated professional programmers. The subjects chosen for this experiment would require a reasonably thorough exposure to scientific and engineering programming.

Also significant is the bias stemming from the computer related experience of the subjects with the time-sharing or batch systems. The experiments done by Erikson at SDC[12] involved subjects who had only on-line experience. On the other extreme Schatzoff, Tsao, and Wiig[16] indicated that their subjects had experience only on batch systems. To obtain subjects for any experiment who would have equal familiarity and skill with both systems is extremely difficult but necessary in terms of experimental control. The subjects employed in this research, as in the IBM study of Schatzoff, Tsao, and Wiig[16], had little knowledge of time-sharing systems. However, practice sessions on the teletype and 100% monitoring of the time-sharing session should minimize the effect of this bias.

The problems used in the various experiments include mathematical problems such as evaluating cosine infinite series and integration, simulation, sort routines, and logic puzzles. While these are certainly not an exhaustive

list, they do represent a fair sample of the scientific programming environment with the exception of statistical analysis. For this research (in the engineering and technical environment in industry) it would be necessary to choose problems of this general type. As Sackman(14) points out, in all the experiments there were no problems involving large data files or processing typical of the business systems environment. Apparently, from an economical point of view, it is not easy to justify experiments requiring large amounts of computer and programmer time. Of course, there are no guides to determine what is normal or representative in the scientific or business arena of programming and computing, but for the limited scope of this research representative problems were developed.

The <u>on-line/off line facilities</u> represent one area where key differences between the experiments are apparent (See Table 2-2). In both Erikson(12) and the Grant and Sackman SDC(5) experiments the programmer was allowed to use the full facilities of SDC's time-sharing system. The data for "batch" mode was gathered by causing a programmer to submit his work to a second party, who then made a time-sharing run in accordance with written instructions he had been given, and then

Table 2-2

COMPARISON OF RECENT & CURRENT RESEARCH: On-Line/Off Line Facilities

EXPERIMENTAL CHARACTERISTIC	SCHATZOFF, TSAO, AND WIIG (16)	ERIKSON (12)	GRANT AND SACKMAN (12)	SMITH (18)	GOLD (14)	JACKSON
On-line Facility	MIT Time-Sharing IBM 7094	SDC Q-32 Time-Sharing	SDC Q-32 Time-Sharing	Burroughs B-5500 batch system at Stanford with "instant" turn-around	MIT Time-Sharing IBM 7904	SDS Mod 940 Time-Sharing thru TTY-33ASR
Batch Facility	IBM 7094 scientific batch facility	same facility simulated off-line condition	same facility simulated off-line condition	same facility with "normal" turn-around	MIT Batch facility, IBM 7094 scientific	IBM 1130 disk operating system
Language Used	not mentioned	TINT interpretative, higher order language for time-sharing	JTS (higher order language) & SCAMP (machine language	Burrough & extended ALGOL	DYNAMO-simulation language used in time-sharing & batch modes	FORTRAN IV in batch & conversational FORTAN IV in time-sharing
Batch Turn-around	not mentioned	usually several minutes, variable	constant at 2 hours	variable, usually several hours	6 hrs. (daytime) 10 hrs. (overnight) variable	controlled to follow a random variable having an exponential distribution with a mean of 2 hrs.

waited until two hours had elapsed (total) before returning the printout to the programmer.

Although the statistics were impressive, the experimental technique is fundamentally fallacious. To simulate a batch operating system as described above changes the environment in an uncertain way. Most batch shops have debugging tools which are devised for the batch environment. The same program deck can be put in several times and several tests can be run back to back with no interference in a normal batch shop. The results of each test can be made completely available both through normal printouts and through rather massive off line dumps. None of these tools is available to the programmer when the batch shop is being simulated by a second party sitting at the teletype.(10)

Further, the raw hardware efficiency of a batch shop is superior to that of a time-sharing facility. Thus, for a straight computational run, the milliseconds of CPU time required to read the input, process it, and write a line of output are considerably less in the batch mode than in the time-sharing mode. However, in the SDC experiment no mention is made of any credits applied for this efficiency phenomenon.

In Smith's study,(18) the basic system was batch. Time-sharing was simulated by providing essentially "instant turnaround", i.e. by providing the capability of having a job run while the programmer waited a matter of a few minutes. This gives the computer availability of the time-shared terminal environment but not the kind of communication that would be used, since most existing time-sharing systems compile a program line-by-line as it is entered at the keyboard and allow syntax corrections during this process.

It is interesting to note that Smith made the following remark concerning the limited simulation of the time-sharing environment with "instant turnaround" batch type processing:

> The fact that a whole program was run at once, allowed the programmer to study his entire program between compilation attempts, thereby giving an opportunity to correct logical errors while eliminating syntax errors. This could be an advantage of instant turnaround over actual time-sharing of the computer; however, at this point we have no evidence to support this conjecture. In other words, correcting syntax errors which are scattered throughout a program gives an opportunity to consider the overall program logic again, whereas correcting syntax one line at a time does not.(18)

It is important that we appreciate Smith's perceptiveness at this point as it will become a cornerstone to the experiment.

Another variable difficult to control in the experimental environment is the computer language used. Both Gold[4] and Smith[18] claimed to have their subjects use the same language equally applicable and useful under both modes. For Smith's experiments this was the Burroughs Extended ALGOL and, of course, since time-sharing was only simulated with "instant turnaround" precisely the same language was used in both, i.e. no conversational debugging tools in the language. It appears that Gold, using the DYNAMO language, had the only compatible software packages which differed only in that one package ran interactively under time-sharing and the other ran under a batch operating system. Otherwise, the two languages were the same, with the same diagnostics, the same formats, same features, and same capabilities. In the research with which this thesis is concerned, the FORTRAN IV language was used under both programming modes. The conversational (interactive) version of FORTRAN IV, however, did vary slightly in format and diagnostics from the batch version. This, I think, is to be expected since:

1. These represent variations typically found between higher level languages on the two systems and
2. Some of the features; debugging messages, format, breakpoint, etc. are features which only have meaning or value in one mode or the other.

Turnaround times under the batch mode in real world situtations vary anywhere from minutes to weeks with perhaps the most common being a matter of a few hours. In the experiments reviewed herein, only Grant and Sackman controlled turnaround at all and they held it at a constant two hours under simulated batch. In Erikson's(12) experiment, turnaround was usually several minutes, while with Gold's subjects it was an average of twice a day. In Smith's experiments at Stanford, turnaround times varied from a few minutes to full days. It is particularly disturbing, for all the studies, that it is not clear whether subject waiting time during batch turnaround time was spent working on the problem. For some of the studies, it is not clear whether it was included or excluded in subject logs of man-hours spent on experimental tasks. It seems somewhat unrewarding to conclude

statistically that, with comparable conditions and long turnarounds, batch takes longer than time-sharing in terms of elapsed time if the programmer is not working on his experimental task until he received his next printout!

The author's research includes a closely regulated turnaround schedule for the batch mode and a clear analysis of allocation of subject's time between turnaround periods.

Table 2-3 gives a summary of the experimental procedures incorporated in the studies being compared. First consider the experimental design. In the exploratory study by Grant and Sackman(5) a 2 x 2 Latin Square Design with repeated measures as shown:

	On-Line		Off Line	
*Group I	Algebra	6	Maze	6
Group II	Maze	6	Algebra	6
		12		12

*Six subjects in each group where each subject does two different problems

Subjects were assigned to the two groups at random, and problem order and on-line/off line order were counterbalanced. The statistical treatment for this design involved an analysis of variance to test for the signi-

Table 2-3

COMPARISON OF RECENT & CURRENT RESEARCH: Experimental Procedure

EXPERIMENTAL CHARACTERISTIC	SCHATZOFF, TSAO, AND WIIG (16)	ERIKSON (12)	GRAND AND SACKMAN (12)	SMITH (18)	GOLD (4)	JACKSON
Experimental Design	Greaco-Latin Squares-4 problem 4 subjects vs. on/off comparison	2x2 Latin Sq. 2 problems vs. on/off comparison	2x2 Latin Sq. 2 problems vs. on/off comparison	matched groups of subjects, each subject taking 2 problems on "batch" and 2 on "instant"	2 matched groups of subjects	2x2 Latin Sq. 2 replicates, 2 problems vs. on/off comparison
Statistical Tests	analysis of variance, correlational analysis	analysis of variance, some nonparametric tests	analysis of variance, factor analysis	descriptive statistical comparison, no tests of statistical significance	a variety of nonparametric statistics comparing the 2 groups	analysis of variance, correlational analysis
Experimental Controls	counter balanced order of experimental design	counter balanced order of problems & experimental variables	biographical items counter balanced experimental order	counter balanced order of "batch" and "instant" modes	questionnaire items, deadline for completed problem	counter balance of order of problems & experimental variables, biographical items
Motivational Controls	not mentioned	trainees class grades	job assignment	class grades	class grades	competition among peers, affect on job assignment
Recording Procedures	computer recording work logs, paper and pencil	computer recording & personal logs	computer records, experimental logs paper and pencil test	computer recording, student logs and questionnaire	computer records, student logs and questionnaires	computer recording, work & experimentor logs, paper & pencil test

ficance of mean differences of performance between on-line and off line conditions and between the Algebra and the Maze problems. There were two analyses corresponding to the two criterion measures; one for programmer man-hours spent in debugging and the other for central processor time. A leading advantage of the Latin Square Design for this experiment is that each analysis of variance incorporated 24 measurements. This configuration permits maximum pooled sample size and statistical efficiency in the analysis of the results--especially desirable features in view of the small subject samples that were used.

Erikson(12) used a similar 2 x 2 Latin Square Design and Schatzoff, Tsao, and Wiig(16) used a Graeco Latin Square model with four problems. In the latter, four programming subjects were selected from technically trained undergraduate students with high programming aptitude. Each individual was assigned an identical set of four programs, two to be coded under time-sharing and two under batch processing. Note that the authors also stated:

> All subjects had some prior programming experience and received a review of IBM 7094 batch processing techniques, a brief orientation on the usage of the IBM 1050 console with the Compatible Time-Sharing System (CTSS) and a summary of the command language for that system.(16)

On the other extreme, Smith[18] used only matched groups of subjects and compared mean scores without any reported measures of dispersion or any tests of statistical significance. After two familiarization problems, four assigned problems were divided equally between Batch and Instant by assigning each student to one of the categories indicated below:

Category	Problem Number			
	1	2	3	4
1	I	I	B	B
2	I	B	I	B
3	I	B	B	I
4	B	I	I	B
5	B	I	B	I
6	B	B	I	I

I - Instant B - Batch (18)

The assignment of these categories was for the purpose of eliminating problems and student bias in the comparison of Batch and Instant.

Certainly with a sample of four subjects the Schatzoff, Tsao, and Wigg[16] study had to have optimal statistical efficiency to demonstrate reliable results; whereas, with Smith's sample of 127 subjects observed mean differences are correspondingly more reliable.

None the less, as Sackman has pointed out, the absence of statistical tests and neglect in reporting measures of dispersion in the data from which statistical tests may be constructed are to be deplored since these practices reduce the cost-effective yield of an experiment, leave quantitative results ambiguous, and deprive the larger community of useful information on individual differences.(14)

All of the experimenters counterbalanced the order of problems and experimental variables except Gold.(4) The experimental vehicle employed by Gold was not really a programming task but a computerized simulation model of the construction industry and its market. The student's task was to formulate and construct a set of decision rules to maximize their profits as an independent, small-scale builder in this simulated cyclical market. The computerized simulation model provided criterion performance scores which constituted feedback for the students by indicating their profit level in response to decision rule inputs for this open-ended problem. In effect, Gold's control, in addition to questionnaire items, was a deadline for the completed simulation problem.

The recording procedures characteristically included computer recording of machine usage, either subject or experimenter logs for man-hours spent on experimental tasks, and questionnaires for recording observations and self-analysis of performances. Also, pencil and paper aptitude tests were used in several of the experiments; the Basic Programming Knowledge Test (developed by USC) for the two SDC studies, and the IBM Data Processing Aptitude Test for Schatzoff, Tsao, and Wiig, and also in the research reported herein. Certainly the computer records were probably the most accurate and unbiased and the logs kept by the subjects were the ones most prone to intentional and unintentional errors. At least in the studies with small numbers of subjects, as in the experiments reported herein, it was easier to keep the subjects under surveillance, to monitor their reporting procedures, and to tactfully resolve discrepancies between them. Neither Smith or Gold with 60 and 127 subjects, respectively, discuss any consideration or bias due to this kind or error.

Table 2-4 summarizes the key performance measures incorporated in each of the exploratory studies. As Sackman observed in his comparison of the five major studies,[14] there are two performance measures which seem basic to all man-hours and computer time required to

Table 2-4

COMPARISON OF RECENT & CURRENT RESEARCH: Measures of Performance

SCHATZOFF, TSAO, AND WIIG (16)	ERIKSON (12)	GRANT AND SACKMAN (12)	SMITH (18)	GOLD (4)	JACKSON
Elapsed time	Debug man-hours	Debug man-hours	Initial program preparation	Problem solving man-hours	Initial program preparation
Analysis	Computer Time	Coding man-hours	Keypunch time	Computer time	Coding man-hours
Programmers Time	Program Size	Computer Time	Time to prepare new run	Task performance	Loading Time
Computer Time	Individual differences	Program Size	Number of runs	Ratings of written reports	Program Size
Number of compilation	Basic Programming Knowledge Test Scores	Program running time	Computer runs per trip	Cost Comparisons	Debug man-hours
Total cost		Individual differences	Elapsed Time	Questionnaire Item	Number of runs
		Basic Programming Knowledge Test Scores	Computer Time		Occurrence of programming errors by cause and detection level
			Submission intervals		
			Questionnaire Items		

complete the experimental task. While computer time is fairly consistent, the man-hours criteria appears partitioned in different ways and in various forms. In the Schatzoff, Tsao, and Wiig[16] study, analysis and programmer time are differentiated in the following way:

1. analysis time- total time in minutes spent by each programmer in programming, analysis and debugging of each problem.
2. programmers time- total time in minutes spent by each programmer on each problem. This includes analysis time plus such items as keypunching and console time.

Gold[4] uses a single measure of problem-solving man-hours while the two SDC studies exclude altogether development, analysis, coding and keypunching and look only at "debug" man-hours. Smith has partitioned man-hours in a similar fashion to the research described herein, i.e. take the total man-hours time and logically dissect that time into the smallest set of unique and exhaustive elements. Because of the differences in definition, cross-comparisons are somewhat difficult.

The questionnaires and ratings of subject impressions were of interest in providing a guideline for user attitudes between the time-sharing/batch modes. Because

they were subject to the problems that plague all questionnaire reliability and validity exercises, little interpretative value can be drawn. Because of the strong bias in historical practice with the batch system, no subjective evaluation of user attitude was attempted in the research described herein.

Only two of the studies (outside that of the author's) recorded size of the final program and running time. Although these are easily obtainable measures, it seems extremely difficult to translate them into an objective measure of program efficiency. The problem chosen for the experiments in all cases did not involve precisely the same degree of complexity. One could attempt to normalize this uncertainty in the absence of accepted measure of difficulty; that is, attempt to get a performance measure by computing CPU time per instruction. This, of course, is one of the most misleading metrics ever devised by man since it rewards the writing of long inefficient code, and thereby, credits the marginal producer with an excellence he does not deserve.

It is also surprising to see that only two of the earlier studies (Gold[4], Schatzoff, Tsao, and Wiig[14]) attempted to estimate on-line versus off line costs

which incorporate both man and machine factors. In both cases these cost estimates were derived from experimental measures of man and machine time which were used as empirical parameters in simple cost models.

Analysis of Prior Results

At this point it is appropriate that we analyze the results and conclusions of each of those experiments discussed. It is on the basis of what they measured and didn't measure that the objectives and justification for this research has been formulated.

In all of the studies under the time-sharing system, programmers required less total elapsed time to complete their experimental tasks. Four out of the five studies show time-sharing (or the simulated equivalent) results in less human time in producing programs. Table 2-5 shows a composite comparison by performance measures for time-sharing versus batch as prepared by Sackman (1967)[14]. Notice the variation in man-hours to complete the desired tasks (from 2:1 in favor of time-sharing to 2:1 in favor of batch mode). This is more striking than the median result for all five studies (a 25 percent advantage for time-sharing).

Table 2-5

Time-Sharing Versus Batch Processing

Experimental Results (Sackman)(14)

	Man-Hours	Computer Time	Costs	User Preference
Erikson	Time-Sharing 1.9:1	Time-Sharing 3.4:1	Time-Sharing	Time-Sharing
Gold	Time-Sharing 1.2:1	Batch 5.7:1	Approx. Same	Time-Sharing
Grant and Sackman	Time-Sharing 1.6:1	Batch 1.4:1	Approx. Same	Time-Sharing
Schatzoff, Tsao and Wiig	Batch 2.1:1	Time-Sharing 1.1:1	Batch 1.5:1	Not Reported
Smith	Instant 1.2:1	Batch 1.5:1	Approx. Same	Instant
Median for All Studies	Time-Sharing 1.2:1	Batch 1.4:1	Approx. Same	Time-Sharing Preferred

It is not even clear how much of the reported man-hour advantage for time-sharing is attributed to the critical elimination of machine turnaround time, to the interactive conversational features of time-sharing, or to some combination of both. Smith's study comparing conventional batch with "instant" batch shows the obvious man-hour savings that go with the faster turnaround. The Erikson[12] study reveals that on-line problem-solving moves considerably faster than batch or non-interactive problem-solving even when turnaround time is measured in minutes.

The comparative results on computer time show no clear trend. The only conclusion that can be made is that computer time is highly sensitive to the unique conditions of each experiment and that no consistent advantage seems to accrue in either mode as far as the combined data of these studies are concerned. It would be unadvisable to conclude anything from the median 1:4:1 ratio in favor of batch mode.

The combined results for man-hours and computer time support the hypothesis that in time-sharing the user is trading off computer time for his own time. Sackman's

analysis suggests that rather than check out his program as thoroughly as he can at his desk, the time-sharing user is more likely to take a less-polished version or only a partially checked program to the computer for a trial run than his batch mode counterpart. Time-sharing critics will assail this practice by claiming that the user develops careless and lazy work habits through excessive reliance on extra computer runs; whereas, the time-sharing advocates will assert that such behavior allows for a more intelligent exploration and testing of alternative solutions at a natural pace for the user when and as problems arise. While there may be some truth to both positions, it is apparent that a key factor in measuring the phenomena would be the number and kinds of errors that the programmer makes and how long it takes him to correct them. If, in fact, the user under time-sharing does develop careless and lazy work habits, one would suspect him to make more errors than his batch counterpart, or at least to take more time (man or computer time) to detect and correct those errors.

If we accept the indication that programming under the time-sharing mode will require more computer time than batch, the question again arises: Is the difference merely attributable to higher machine efficiency under

batch processing (more efficient throughput), or is it also associated with more exploratory trial and error human behavior under time-sharing? Are the users really more careless in preparing and desk checking their work when they have on-line access to the computer? The study reported herein has attempted to gather some comparable data on human error rates under both modes.

Is it altogether clear that less computer time per task is always desirable because (other things being equal) it means lower cost? Other things may not be equal if the more aggressive user knows how to exploit the services of the computer system by using more, rather than less, computer time to develop higher quality finished products with the increment in quality offsetting the expense of more computer time.

No clear advantage for time-sharing or batch is evident from the cost comparison. The two studies which did indicate a cost difference (in opposite directions) had uncontrolled conditions which biased the results; such as, subjects experienced only in batch methods, or a language constructed specifically for time-sharing. Obviously, the results are contingent upon unique experimental conditions. They do not disagree with the idea of a programmer-

computer tradeoff, i.e. as man-time cost goes up, a hypothetical breakeven point is reached which dictates the operation of a user oriented configuration (minimizing man-time) at the expense of computer time; hence time-sharing mode. Again, these comparisons are obviously based on doing particular tasks, not long range assignments.

Also Sackman's analysis shows that user attitudes tend to favor time-sharing or, as in the case of Smith, the "instant" batch alternative. It is hardly surprising that experimental subjects liked the easy access to computers and fast computer response. Nevertheless, the response might well be a function of the experience level of the programmer and the programming mode with which he has been traditionally oriented. Note that the Schatzoff, Tsao, and Wiig study, which did not report any user preferences, was the only experiment involving experienced programmers with the batch mode. The experiment reported herein, involving experienced programmers with batch mode experience of the fast turnaround type, offers an alternative hypothesis.

The five experimental studies typically made provision for guaranteed access to the time-sharing mode with comparatively normal access and uncertain turnaround

time in the batch mode. In addition, most studies were conducted at centers that pioneered in time-sharing using subjects who tended to have favorable predispositions toward the charms and attractions of time-sharing.

Section 3

Problem and Experimental Design

In the discussion section of Gold's[4] thesis, a hypothesis appears which states that perhaps instantaneous interaction from a time-sharing system terminal is not universally good. For instance, he indicates that perhaps turnaround time can be too fast since people are inclined to try another run before they have completely analyzed the previous run.

Let me state this idea somewhat more formally:

An overview of program construction and particularly the relationships of program modules is seriously impaired with the incremental compilation and instant turnaround characteristics of conversational programming on the time-sharing system.

I have chosen to call this concept the Principle of Forced Interaction--forced interaction because the programmer is in a sense at the mercy of the system and its interactive characteristics.

In order to experimentally investigate the existence of the Principle of Forced Interaction in terms of the physical significance and its effect on

cost/effectiveness of time-shared systems compared to batch, one must certainly develop suitable measurement criteria.

The effect of the principle could be felt in only two directions--the time required to complete the task (which should be longer if the subject is guilty to oversight of relationships between program modules, for example) and the errors involved in solving the programming task.

Time to Complete Task

Let us consider the time required first. Most of the studies reported herein[4,12,15,18], comparing conversational time-sharing programming with batch processing, have shown that time-sharing requires less net or human time than batch for the same job. The performance measures used by the five experimenters to capture the net man-time varied considerably, as pointed out in the background section. The comparative table of results, as compiled by Sackman[14], shows that in four of the five experiments time-sharing (or its equivalent) resulted in less human time in producing programs (1.2:1). This would suggest that either the Principle of Forced Interaction does not exist or its effect cannot be translated into time.

The explanation lies in the latter and the effect on time of the Principle of Forced Interaction can be illustrated by studying the debugging process. To do so, we must consider the debugging process in three segments:

1. diagnostic syntax checking
2. execution error response
3. system and logic - no system aid

Diagnostic Error Checking

Under conversational time-sharing, in addition to providing instant turnaround, there is line by line compilation of the program text as it is entered in the keyboard, and there is corresponding line by line diagnostic syntax checking. Thus, if a significant portion of the errors normally made in program development are of the syntax detectable type and since the time (under conversational time-sharing) required to diagnose and correct these from the keyboard approaches zero, there is a dramatic reduction in net time requirement.

Execution Error Response

For debugging at execution time, both the conversational time-sharing and the batch systems have

extensive error checking and corresponding error messages. Assuming for the moment that these are equivalent, let's examine what happens with a typical execution type error and the effect on the time required to resolve it.

Under the batch system, first a cursory investigation of the program logic and flow based on the diagnostic message test is performed. If the source of the error is not revealed, then perhaps hypothetical test data are carried through the program calculations to check reasonableness and/or correct logic processing. If the source of the error is not yet discovered, a last resort might be to insert in the program some output statement to show the value of working variables. This attempt to follow the program by stages should determine what modules performed correctly and at what point in the program's execution difficulty was encountered. Obviously, it affords no opportunity to look at the correctness of modules beyond that one in error.

Conversational time-sharing systems include capabilities variously entitled: BREAKPOINT, PERFORM, PROCEED, etc., which allow the user to temporarily halt execution at specified points in a program.

When a BREAKPOINT is reached in the program, execution halts prior to the BREAKPOINT statement. The statement is printed out and control is transferred to the command mode so that the user can either take debugging action or proceed. The PERFORM command allows a user to execute a statement without having it permanently compiled into the program. The command may only be given during a halt in execution caused by a BREAKPOINT. The PERFORM command is used in conjunction with the BREAKPOINT command for debugging purposes. For example, the user can display values at various breakpoints to examine the changing values of variables being manipulated by the program. If he discovers an error in the computation of a variable, he can perform a REPLACEMENT statement which sets the variable to the expected value and he can then proceed to check out the rest of the program before following up on the previous error. Since statements inserted with the PERFORM command do not become a permanent part of the program, the user is relieved of the duty of deleting them. Thus obviously, for a portion of what we have referred to as execution time errors, conversational time-sharing will offer faster diagnosis and isolation.

System and Logic - No System Aid

The third general area of debugging relates to the time lost in detecting and correcting for errors in logic and flow between program modules, and those requiring an overview of the program construction. These intuitively seem to be the most difficult to detect since the system offers no aid to the programmer in identifying these kinds of errors. These are the types of errors that the Principle of Forced Interaction claims are more difficult to detect and diagnose under time-sharing than batch.

Sitting at a time-sharing terminal, the user is concentrating his attention on the ten or twenty statements of the program test that appear before him. It is very time consuming and cumbersome for the user to repeatedly look at updated listings of his program files. In making a correction or an addition, the user may even employ a variable name or branch to a location already used and thus actually propogate an error requiring more time to correct.

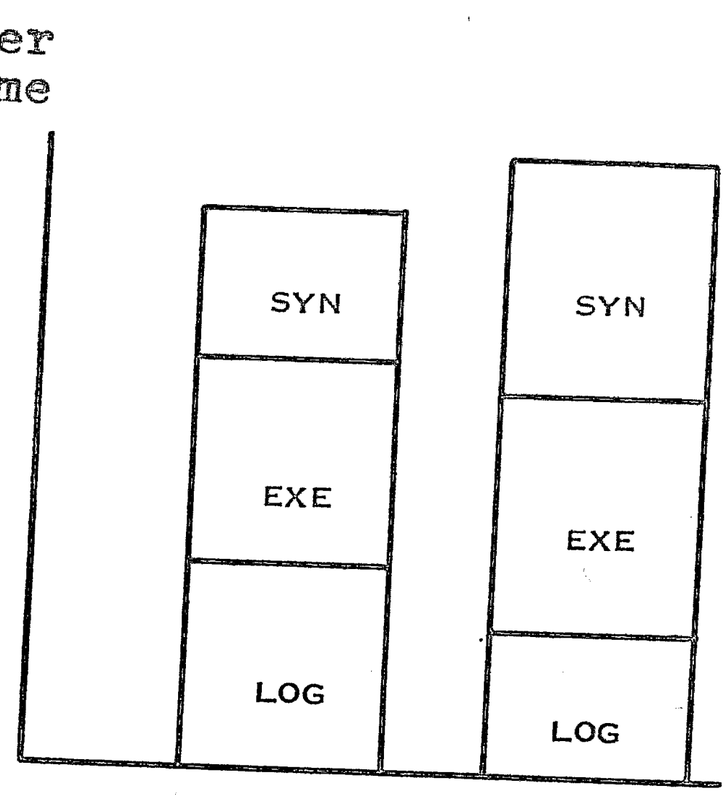
If the Principle of Forced Interaction exists and has a measurable effect on time measured and yet the overall time required is less in conversational time-sharing systems than in batch, the following kind of relationship may be true:

let T_i = total time

where $i = \begin{cases} TS - \text{Time-Sharing} \\ B - \text{Batch} \end{cases}$

let t_{ij} = time required or segment of debugging

where $j = \begin{cases} SYN - \text{syntax error correction} \\ EXE - \text{execution error correction} \\ LOG - \text{logic and system error correction} \end{cases}$



The experimental evidence indicates

$$T_{TS} < T_B$$

so if the described decomposition of debugging is complete

$$t_{TS,SYN} + t_{TS,EXE} + t_{TS,LOG} < t_{B,SYN} + t_{B,EXE} + t_{B,LOG}$$

If, as has been asserted by the author in the Principle of Forced Interaction,

$$t_{TS,LOG} > t_{B,LOG}$$

then

$$t_{TS,SYN} + t_{TS,EXE} \ll t_{B,SYN} + t_{B,EXE}$$

Programming Errors

Now that we have explored the effect that the Principle of Forced Interaction has on time required to

complete a programming task, we must obviously look next at programming errors as our other source of measurement. What constitutes a programming error? How can they be classified? How does one measure the seriousness of one error compared to another? Are there errors peculiar to only one of the programming systems--batch or time-sharing?

Webster says of an error: "...something incorrectly done through ignorance, carelessness, inattention or misunderstanding and does not in itself carry a strong implication of criticism". What is the criticism (penalty) for making an error in designing and debugging a program? The penalty involved is a function of the amount of time and effort to have the program be entirely correct or at least give correct answers (a distinction which will be elaborated upon later).

The author had the experience of teaching an introductory level of FORTRAN IV course to a group of undergraduates prior to undertaking this analysis. Through working with the students and monitoring their progress, a classification scheme for programming errors emerged. Two distinct classification entities seemed clear:

1. Type of Error by Cause - This identified the cause for the error's occurrence, i.e. did

the subject make a mistake by forgetting the rules of programming, or did he mispunch a card when entering his coded program file?

2. Error Detection Level - This gives a measure of the seriousness of the error in terms of the effort required to detect and correct it. For example, is it the kind of error for which the compiler will detect and issue a message or will the program compile and execute to completion but give incorrect answers?

Such a classification of error characteristics clearly is necessary because it has already been demonstrated that some features of the conversational time-sharing system save time compared to batch in error detection and correction. Programming errors corrected using these features will be done quickly. If the Principle of Foreed Interaction hypothesis is true, then those errors (by the classification scheme) affecting the logic and flow of information between modules will take a longer time or many tries to correct under time-sharing.

The next step was to develop levels and corresponding definitions for each of the classification. Table 3-1 shows the classes and interpretations that were developed

Table 3-1

Causes of Errors in

Program Development

Misinterpretation

to conceptually design a sequence of programmable operations which will not functionally duplicate the steps to be performed based on the problem definition. Includes non-recognition and omission of operations to be performed as well as substitution of incorrect operations.

Transformation

to have correctly interpreted the sequence of operations to be performed, but to implement program statements which will not execute the desired operational sequence; to have the desired or intended operational sequence not execute interpreted procedures in the sequence of programmable statements resulting was incorrect.(For example, by sorting a list in ascending sequence when descending sequence was desired)

Entry

to error in entering data, program control, or program statement information to the computing system; and the desired sequence will not execute because of the entry of such program information.
It may occur from the following types of data entry mistakes:

omission
transposition
substitution
insertion

from any of the following points in the program development system:

written on coding sheet
key punch
entering into teletype terminal

for Error Cause. It should be noted that considerable work has been done in the area of ENTRY error analysis(19), less work in TRANSFORMATION type errors, and to the author's knowledge, no research has been done on MISINTERPRETATION type errors.

In attempting to define levels and definitions for the Detection Level classification, two facts became apparent:

1. There is an implied hierarchy in detection levels.
2. Detection level classifications may be segregated into two distinct groups:
   a. Those where the system aided the user in detecting, isolating, and diagnosing his error.
   b. Those where the system provides no such assistance, i.e. the programmer is on his own.

Table 3-2 shows the five classifications for detection level and the corresponding definitions. Note that Logic and Extended or Logical Incompleteness are primarily differentiated by the fact that in the latter, a test set of data will yield correct answers but that for some acceptable sets of data it will not. Since this scheme

Table 3-2

Program Error Detection Level

System Aided Detection

Diagnostic - Compiler-Detected

Those errors that will be detected with the compiler and denoted by a diagnostic message or flag (for example, an uneven number of left and right parenthesis, etc.) - syntax checking.

Diagnostic - Execution-Rejected

Programming errors not detected by the compiler but whose existence will preclude the program from executing to completion (for example, program in tight loop).

Programmer Detected

System (Expression)

Error of program construction confined to an expression or an operation which either will not perform the desired operation or will perform other than the desired operation (the flow chart operation module)

Logic

Error in the logical flow or transfer (branching) of program control, i.e. sequence, conditional, tests and branches on a flow chart either where there is no branch or an incorrect branch.

Extended - Logical Completeness

Error that will not inhibit the execution of the program under a normal set of test data but will not permit exhaustive results (data or analysis). This means that data representing some extreme conditions, while valid, will not work due to said error condition.

has been hypothesized as a cross-classification, we can represent the classes in matrix form as in Table 3-3.

The first question that naturally arises in considering the proposed error classification scheme is obvious. Can one follow the subject developing and debugging a problem either under batch or time-sharing and using the proposed classification, classify <u>every</u> error encountered? In other words, are the classes mutually exclusive and exhaustive? This validation was carried out in two ways:

1. Analysis of documented results of student FORTRAN programs from undergraduate introductory courses
2. Analysis of results from administering a set of test programming problems to selected graduate students in a Process Capability Study. (See Appendix 8)

The results of applying the classifying scheme indicated a clear classification (more than 90% of the errors committed) was possible if the analysis of the documentation-flow chart, and coding sheets and listings was followed up with a personal interview with the subject. If an error was not easily classified as to cause, the following questioning process was used:

Table 3-3

Error Classification - Matrix Structure

PROGRAMMING ERRORS

CAUSE - DETECTION LEVEL MATRIX

DETECTION LEVEL / CAUSE	SYSTEM AIDED		PROGRAMMER DETECTED		
	DIAGNOSTIC COMPILER DETECTED	DIAGNOSTIC EXECUTION REJECTION	SYSTEM (EXPRESSION)	LOGIC	EXTENDED - LOGICAL INCOMPLETENESS
MISINTERPRETATION					
TRANSFORMATION					
ENTRY					

1. Point out error in program to subject and state detection level.
2. Interrupt subject immediately and ask: "Did you mean to do ....(nature of error)?"
3. If subject response was no, then cause is ENTRY type.
4. If subject response was yes, then ask: "What did you want to do?"
5. If subject response was correct operation, then error is TRANSFORMATION type.
6. If subject response was incorrect operation, then error is MISINTERPRETATION type.

A programming error classification scheme has now been developed which would permit investigation of that type of error effected by the Principle of Forcer Interaction. It was next necessary to develop some measures of efficiency or effectiveness in terms of error classifications.

Measures of Effectiveness for Errors

The following measures were considered significant to measuring the severity or impact of program errors on system performance by classification.

1. Total Number of Errors - Simply the absolute magnitude of errors occurring without regard to how long they lasted or how much effort was required to detect them. Most errors in programming are imbedded in the program code by the time the first execution is requested. The number of errors should reflect the "error proneness", and thus a measure of performance in the programmer.

Let $e_{ijk} \equiv$ an error of the i^{th} cause and the j^{th} detection level and the k^{th} error in that group

$$= \begin{cases} 1 \text{ for } i = 1,2,3; \quad j = 1,2,3,4,5 \\ 0 \text{ otherwise} \end{cases}$$

$\therefore$ Total number of errors $E_{ij} = \sum_{i\ 1}^{k} e_{ijk}$

2. Maximum Error Run Units - Of all the errors that occurred in the classification, the maximum error run is the largest number of executions (runs) of the program it took to detect and correct the worst of the errors in that group.

Let $r_{ijk} \equiv$ the number of executions (runs) required to detect, diagnose and correct error e_{ijk}

$\therefore$ Maximum error run units $R_{ij} = \max \{r_{ijk}\}$

This measure is perhaps more reasonable than just the total number of errors because it measures the limiting amount of work necessary before the program is complete and correct. It, therefore, represents a worst case design.

3. Error Reduction Rate - This is a ratio which measures the relative duration of existence, or "life", of a group of errors (again in terms of the number of executions or runs that were required to detect, diagnose and correct). The ratio is formed by the quotient of the number of errors in a classification divided by the sum of the error run units associated with those errors.

$$\therefore \text{Error Reduction Rate} = \text{ERR} = \frac{\sum_{1}^{k} e_{ijk}}{\sum_{1}^{k} r_{ijk}}$$

Thus, if every error is corrected (killed) immediately after the execution in which it is committed (born), then the error reduction rate (ERR) will be at its upper limit of 1. As the number of executions required to detect, diagnose, and correct the errors increase, the ERR decreases and asymptotically approaches 0.

Thus, a high error reduction rate indicates a high efficiency in error correction or debugging. This measure may, however, yield misleading results because it doesn't consider the range of variation in the error run units.

As an example, consider the classification System - Transformation where the following data are recorded for three experiments.

Experiment	Error K	No. Error Run Units r_K	Total Basic Errors E	Max. Error Run Units R	Total Error Run Units	Error Red. Rate
1	1	3	3	15	21	.125
	2	6				
	3	15				
2	1	8	3	8	21	.125
	2	8				
	3	8				
3	1	15	2	15	16	.125
	2	1				
	3	1				

Obviously, results of any experiment will have to consider each of these measures in combination with the others, and thus all were included in the experimental procedure.

Additional Performance Measures

In addition to measures of efficiency concerned with program error correction, several additional measures were considered as employed by the former experiments and outlined in Section 3. These included the following:

1. System and Flow Charting Time - This is the net time devoted to the conceptualization of the problem and design flow charting of its solution.
2. Coding Time - Time required to code the problem from the flow chart.
3. Loading Time - Net time required for key punching, if performing under the batch mode; and terminal loading, if under the conversational time-sharing mode.
4. Debugging Time-Net - The net time (in minutes) spent in debugging from the time the loading is complete to the time the program is complete and correct. Time spent during this period on other tasks has been excluded.
5. Debugging Time-Elapsed - The elapsed total clock time (in minutes) from beginning of program debugging to program complete and correct.

6. Program Efficiency - A measure of the efficient use of coding in program construction determined by the number of statements in the program less all comments and I/O commands.

7. Total Number of Runs - Total number of executions until a complete and correct solution was realized.

Section 4

Procedure

In establishing an experimental procedure the following areas had to be investigated and resolved:

1. development and validation of test problems.
2. acquisition of an environment and subjects with which to conduct the experiments.
3. development of a specific procedure in which the experiments would be performed and the data collected.
4. establishment of statistical procedure through which an analysis of the results could be derived.

Experimental Problems

In the development of the experimental problems, the following characteristics were desired:

1. Problems which would be representative of the realm of programming tasks typically encountered in the environment being simulated. The environment chosen for this experiment was the small engineering/technical/research department which implies a program mix favoring technical scientific tasks over business data processing tasks.

2. Problems which would be of such a nature so as to provide opportunity to involve most of the capabilities of a scientific programming language such as FORTRAN. Thus, the subject would be able to exercise his command of the programming language to achieve the desired results--an efficient, computationally correct program in a minimum amount of time.

3. Problems which could be solved in a relatively short time--a matter of hours. This constraint existed mainly because of practical and financial limitations in the experimental environment.

4. Problems so constructed to elicit a high probability of occurrence of each of the types of errors by cause and detection. That this characteristic can be influenced by program design is clear when one considers logical incompleteness or extended errors (for example, using a computer to start at zero so that one more record than the file has room for is executed if maximum file data is given).

The idea of developing programs that contained imbedded errors was considered first. That idea was rejected because it was felt that such a procedure was so artificial and atypical of programming practice that it would invalidate the experiment. It was decided, therefore, that each subject would:

1. code his own program
2. use his own logic

Four problems were developed with complete write-ups and each appears in Appendix 8. The problems were:

1. Finding the Root of a Polynomial with the Newton-Raphson Method.
2. Simulation with Monte-Carlo Sampling of Stochastic Inventory Model.
3. Solving a system of simultaneous equations by the Gauss-Siedel Method.
4. Data processing program for calculation of customer invoices for utility services.

A Process Capability Study was conducted with these four problems to evaluate the experimental procedure. This involved the selection of graduate students with substantial programming experience. They were asked to develop one of the programs; two of them performed under

batch mode and two of them performed under conversational mode. No attempt was made at this time to evaluate other environmental factors such as language simularities or turnaround time because the objective here was to determine if the proposed experimental procedure was feasible.

The Process Capability Study showed three things:

1. If the subject was unfamiliar with the mode of program development, say a user with batch experience operating under time-sharing system, questions would arise which if not answered would cause errors attributable solely to the subjects inexperience with the systems procedures. For example, "how to display a range of statements from the terminal". If the subject was monitored 100% of the time, these questions could be answered immediately with no appreciable loss of time and without the subject committing an error. Obviously, what was happening was that the subject was still on the learning curve and only through 100% monitoring could its effect be eliminated. This practice was carried out for all of the experiments.

2. The two problems which seemed to be most representative and yet seemed to evoke the widest distribution of programming errors were (2.) Simulation with Monte-Carlo sampling of stochastic inventory model and (4.) data processing programs for calculation of customer invoices for utility service. These were chosen as the experimental problems.

3. Analysis of the documentation, namely flow charts, coding sheets and all program listings along with an interview with the subject (employing the questioning procedure) revealed that a very high percentage of the errors committed could be classified clearly into a cause and detection level. This was particularly gratifying and suggests that the classification scheme may have some validity.

It was also apparent from the Process Capability study that with problems which require no more than a few hours to flow chart and code, the number of errors propogated by the subjects will not be sufficient to always fill out the classification matrix. Obviously and unfortunately, this would inhibit the analysis of the data.

The Experimental Environment

The environment chosen to conduct the experiments was the small engineering, research, education or manufacturing center; that is, the type of environment where the choice between a small in-house stand alone computer versus a terminal connected to an outside conversational time-sharing service is considered. The site chosen for the experiments was the Western Electric Graduate Education Center at Princeton, New Jersey. Here approximately 15 outstanding and aspiring young men from Western Electric locations are chosen each year to undertake graduate studies leading to a Masters of Science degree in Industrial Engineering. The subjects chosen for the experiment were all members of this program and all had at least a few years of industrial experience. The subjects (eight of them) represented a fairly homogenous group in the following ways:

1. They all had superior academic backgrounds--greater than 3. out of 4. cumulative undergraduate averages.
2. They all had a minimum of three years industrial experience with Western Electric.
3. They were all presently second year students in the graduate program in Industrial Engineering.

4. They all had a minimum of six months experience with FORTRAN programming and had a formal FORTRAN class instruction.

Numerous other statistical data including test score results were also collected on the subjects which will be presented in the results section.

Facilities

At the education center for the batch system, an IBM 1130 configuration was used. This included a card reader punch, an 100 line/minute printer, a disc pack for program and data storage and three keypunches (029). A version of FORTRAN IV, available on this system, included most of the standard FORTRAN capabilities. Included in Appendix 11 is a complete listing of the diagnostic messages available on that system. The environment for use of this computer was described as "open-shop and hands-on". This means that the subjects run their own jobs when they are ready. The turnaround, obviously, is much smaller here than in most large computing centers, however, it should be representative of this type of environment. Delays, when they do occur, are a result generally of a queue of students waiting to use the computer. With its rather slow I/O gear, a small wait is not infrequent.

As was pointed out in the literature search, the turnaround time has not been well monitored in previous experiments. In this experiment, the turnaround time was set to follow an exponential distribution with an expected value of one hour. Exponential values were generated and the subjects were asked to adhere to them strictly. Thus, most runs were returned in a matter of minutes and occasionally not for a few hours. The exponential distribution of turnaround times the subjects felt would be particularly representative of this type of computing environment.

The time-sharing environment was accomplished by having a terminal Teletype 33KSR connected with a Dataphone coupler to a commercial time-sharing service, use of an SDS 940 computing system and offering a conversational FORTRAN IV language. Included in Appendix 10 is a complete listing of the diagnostic message available on that system. Notice that both the IBM 1130 FORTRAN and the SDS 940 conversational FORTRAN have very similar syntax checking capabilities. The terminal was available for subject use with no turnaround or delay and, as the experiments were staggered over a three-week period, it was possible to provide 100% monitoring of the subjects activity.

Procedure for Conducting Experiments

The experiments were conducted, as designed, during the month of August, 1968, at the Western Electric Graduate Education Center in Princeton, New Jersey. As previously noted, this was an attempt to duplicate the computing environment typically encountered in the small engineering, research testing, and development shop or division. The Western Electric Graduate Education Center was ideally suited in the following ways:

1. The facility was essentially isolated from the main research labs of Western Electric and housed a batch processing IBM 1130 computer with Disc Operating Systems.
2. The students, who were chosen as subjects, were members of this graduate education program and each had several years experience in various engineering and technical disciplines in the Western Electric organization. (There entrance into the graduate program was based on their high academic talents and a demonstrated superior ability in their manufacturing oriented previous jobs.
3. All of the subjects were mature, as well as intelligent, interested in contributing to the

authors experiments, cooperative and yet in knowing each other--competitive (which by the way, provided one of the motivational controls).

In any system where user demand may exceed system capacity, guaranteed access for any group of users may interfere with others who do not have such access. The number of subjects to participate in the experiment at any one time, therefore, was limited to two.

As it turned out, indeed, this was a fortunate occurrence because it facilitated 100% monitoring of the subject. As an example of the significance of this, consider the subject at the teletype for time-sharing trying to interactively debug his FORTRAN IV program. With all of the subjects, prior experience had been limited to batch processing systems. Thus, time-sharing systems were not only new to them, as far as the procedures or mechanics, but those techniques or features of a conversational system, i.e. print debugging and partial execution with breakpoint statements represented really "modus-operendi" for program debugging with which they just couldn't adopt overnight.

As a result, the following two steps were taken:

1. Before the subject started on the assigned task, a minimum of two hours instruction was given on the use of conversational FORTRAN IV with the time-sharing system. The subject actually coded and loaded some sample problems with errors to familiarize himself with the unique error detecting and correction procedures available to him with conversational time-sharing.

2. During every subjects sessions with the time-sharing terminal, the author provided 100% monitoring. At any time when the subject had a question regarding the mechanics of operating the time-sharing system--a question that would not have existed if he had more experience and familiarity with the time-sharing system--then the answer was supplied to him immediately. Appendix 6 shows a documented session of questions and answers that occurred with one of the subjects under this mode of operations. Notice that by these two steps we have quite realistically simulated the conditions (and thus

will measure and analyze the performance) of subjects who are experienced with both batch processing and time-sharing systems.

Starting with the first two subjects, the two problems were then assigned one to be done on the time-sharing system and the other by the other subject on the IBM 1130 batch system. Appendix 8 shows a description of the two problems as they were handed to the subjects. Note that for the Calculation of Utility Invoice problems a file of customer records (20) was either provided in card deck form (for batch) or loaded into disc storage for recall at the terminal (for time-sharing).

For the subject who was currently working on a problem via the batch method, instructions were given to follow strictly a prescribed set of turnaround times, i.e. from the time the subject initiated the run on the IBM 1130 until the time that he first gazed at the results. This prescribed list of turnaround times was designed as a random variable following an exponential distribution with a mean of exactly one hour. The procedure for generating these numbers and a list of the determined value may be found in Appendix 9 - Batch Turnaround Times - exponential with mean of one hour. In choosing a turn-

around time of one hour average, following an exponential distribution, one realized that this is shorter than commonly encountered in many off line batch systems; and yet it was long enough so that most of the subjects complained about waiting. It does not seem unusually short for an engineering shop with a "hands-on" small scientific computer.

Figure 4-1

Page of

PERFORMANCE–TIME RECORD

Program	Subject	Mode

WORK ELEMENT	OBSERVED TIMES				WORK ELEMENT	OBSERVED TIMES			
	START	STOP	ELAPSED	DATE		START	STOP	ELAPSED	DATE

In order to record the time (net and elapsed) that the subject spent on the various work elements of system design, coding, debugging, etc. the form Performance Time Record shown here was developed. Notice that it includes time spent by the subject when he was not involved in work with the experimental task. Obviously this provided an arithmetic check on the accuracy with which the subject filled out the form, but more importantly (in the case of batch) provided a chance to investigate the subject productivity during the time interval between runs.

WORK ELEMENTS		1. System and flow chart 2. Coding 3. Terminal–loading program 4. Executing and Debugging terminal 5. List–debugging–batch		6. Keypunching 7. Seeking outside aid 8. FORTRAN language assist or review 9. Non-experiment time 10.

When the subject completed the problem, he would submit all copies of runs (if on the batch process) or the entire teletype log (if on the time-sharing). The next process immediately to follow was the detailed analysis of the listing to:

1. identify the existence of
2. classify by cause and detection level
3. determine the life of

for every programming error committed. This included any errors that might have been propogated (born) inadvertantly as a result of correcting (the death) of another error and those (such as logical incompleteness errors) that may have never been discovered when the task was submitted as complete. Appendix 1 shows a set of documentation and error analysis for an experiment under batch and one under time-sharing. Remember the identification of the error classification and detection level required the post-experiment subject interview in most cases.

Figure 4-2

PROGRAM DEVELOPMENT–ERROR ANALYSIS		Page of
PROBLEM	DATE	SEQUENCE
SUBJECT	MODE	ELAPSED PERIOD

NUMBER	CAUSE AND/OR DETECTION LEVEL	EXPLANATION AND OBSERVATIONS	RUN NUMBER

As the Process Capability Study indicated, another form was needed to capture all of the error analysis information as the documentation was analyzed (a sample of which is shown here entitled Program Development-Error Analysis).

Statistical Design of Experiment

The statistical design for the experiment used a 2 x 2 Latin Square model with four replications. Eight subjects were each given the two problems, the inventory simulation and the customer invoice preparation. One problem they would prepare under conversational time-sharing and the other problem under batch processing. The sequence was reversed with every other subject so as to balance the design. Also, the order in which the problems were given was switched for the second four subjects. The 2 x 2 Latin Square model, assuming the requirements for analysis of variance were upheld, would allow one to segregate the total variance in the results into four groups:

Variance Attributed to	Degrees of Freedom
1. The programming mode-time-sharing vs. batch	3
2. The problem--inventory simulation or customer invoice preparation	3
3. The difference between subjects confronted with the interaction between mode and problem	3
4. The residual error	6

We must review the concepts of this statistical analysis tool and assure ourselves that the conditions it requires and assumptions it makes are in fact met.

Discussion of Fixed 2 x 2 Latin Square Experimental Design

In general, let us suppose that the three factors of a Latin Square have r levels each and that the observations are $X_{ij(k)}$ where i, j, k 1,2,....r, and where i refers to rows, j to columns, and k to letters in the square. The (k) enclosed in parenthesis to indicate that it is not independent of i and j. The observations are assumed to come from normal populations with the same variance σ^2 and with means

$$\xi_{ij(k)} = \xi + \alpha_i + \beta_j + \gamma_k$$

in which $\sum \alpha_i = 0$; $\sum \beta_j = 0$, $\sum \gamma_k = 0$. All interactions are assumed to be zero in this model.

If we denote the row means by $\bar{x}_{i\cdot}$, the column means by $\bar{x}_{\cdot j}$ and the means of the observations associated with the k^{th} letter in the square (the k^{th} level of the third factor) by $\bar{x}_{(k)}$, the sum of the squares may easily be partitioned as follows:

$$\sum_{ij} \left(\bar{x}_{ij(k)} - \xi_{ij(k)} \right)^2 = r \sum_i \left(\bar{x}_{i\cdot} - \bar{x} - \alpha_i \right)^2 + r \sum \left(\bar{x}_{\cdot j} - \bar{x} - \beta_j \right)^2$$

$$= r \sum_k \left(\bar{x}_{(k)} - \bar{x} - \gamma_k \right)^2 + \sum_{ij} \left(\bar{x}_{ij(k)} - \bar{x}_{i\cdot} - \bar{x}_{\cdot j} - \bar{x}_k - 2\bar{x} \right)^2$$

$$+ r^2 \left(\bar{x} - \xi \right)^2$$

All these sums on the right are independently distributed by chi-square laws (on division by σ^2), the various sums have degrees of freedom indicated in the following ANOVA table.

Source	Sum of Squares	Degrees of Freedom
Rows	$r\sum(\bar{x}_{i\cdot}-\bar{x})^2$	$r-1$
Columns	$r\sum(\bar{x}_{\cdot j}-\bar{x})^2$	$r-1$
Diagonal	$r\sum(\bar{x}_{(k)}-\bar{x})^2$	$r-1$
Residual	$\sum(x_{ij}-\bar{x}_{i\cdot}-\bar{x}_{\cdot j}-\bar{x}_{k}-2\bar{x})^2$	$(r-1)(r-2)$
Total	$\sum(x_{ij(k)}-\bar{x})^2$	r^2-1

The three null hypothesis $\alpha_i=0$, $\beta_j=0$, $\gamma_k=0$ are then tested by dividing the appropriate mean square by the residual mean square and using the F distribution. An analysis of variance program was written in FORTRAN IV to make these calculations.

Assumptions of ANOVA - Bartlett's Test

The statistical model we have developed involving the analysis of variance technique assumes the following form:

observed value = $\sum$ (parameters representing assignable effects)

$+\sum$ (random variables representing assignable effects)

$+$ (random variables representing unassignable (residual) effects)

It is a matter of experience that the more factors we introduce the less the unassignable (residual) variation which is left unaccounted for, but there is nearly always

some such residual variation remaining. The residual variation is suitably represented by some form of random variables, one for each observed value. The analysis of variance is based on certain plausible assumptions about these random variables.

1. The expected value of each residual random variable is zero. This means that all variation in expected values is taken care of by the parameters representing assignable effects ... Restating E(xij) O for all i and j.
2. The residual random variables are mutually independent. This assures that there is no link between different observations which is not accounted for by the terms representing assignable effects. If individuals are chosen at random and measured separately, as was the case with this experiments subjects, the assumption is quite reasonable.
3. The residual random variables all have the same standard deviation. This "homoscedasticity" is really a critical assumption. Stated in another way: Var $(X_{ij}) = \sigma^2$ for all i and j. This condition should be satisfied (at least approximately) for standard analysis of

variance methods to be applicable. One way of determining whether a set of "g" variances is homogenous is to use the quantity:

$$M = 2.3026 \left\{ n \log \sum n_1 s_1^2 - \sum n_1 \log s_1^2 \right\}$$

where the s_1^2 is the independent estimate of variance to be compared, based respectively on n_1 degrees of freedom, and where $n = \sum n_1$. M is proportional to the log of the ratio of the arithmetic mean to the geometric mean of the g sample estimates of variances and is approximately distributed as χ^2 with g - 1 degrees of freedom. M. S. Bartlett(3) has shown that if C is defined by the equation:

$$C = 1 + \frac{1}{3(g-1)} \left\{ \sum \frac{1}{n_1} - \frac{1}{n} \right\}$$

then M/C is somewhat more closely approximated by a χ^2 distribution with g - 1 degrees of freedom than M alone.

Itmust be remembered that the above test is based on the assumption that the random variation within each of the g groups follows the normal. If this is not true, a significant value of M or the ratio may indicate departure

from normality rather than heterogeneity of variance. "A test of this kind is, indeed, more sensitive to departure from normality than the ordinary tests of the analysis of variance."(3)

Other Tests - Aspin-Welch

Another possible statistical test one might consider would be the Aspin-Welch test for testing the difference between two sample means when σ_1' and σ_2' are unknown and may not be equal. Surely this test is much weaker than using analysis of variance in our Latin Square model because it considers variance in simply the two "samples", i.e. the observations made under the batch environment and the observations taken under the conversational time-sharing environment. No statistical consideration is given thus to the strong variation among subjects and variation in the requirements of the two different programming problems. The sample illustration to follow should certainly verify this argument:

The Aspin-Welch test consists in treating

$$t = \frac{d - d_0}{\left(s_1^2/N_1 + s_2^2/N_2\right)^{1/2}}$$

as if it had a t distribution with degrees of freedom given by,

$$n = \left\{ \frac{1}{\frac{c^2}{N_1 - 1} + \frac{(1 - c)^2}{N_2 - 1}} \right\}$$

where $$c = \left\{ \frac{s_1^2/N_1}{s_1^2/N_1 + s_2^2/N_2} \right\}$$

and where $$s_1^2 = \frac{\sum (x_1 - \bar{x}_1)^2}{N_1 - 1} \text{ and } s_2^2 = \frac{\sum (x_2 - \bar{x}_2)^2}{N_2 - 1}$$

The interesting thing about the Aspin-Welch test is that it yields a test with degrees of freedom at least as large as those when we know $\sigma_1' = \sigma_2'$ and run the ordinary t test:

$$t = \frac{\bar{d} - d_0}{s\left(1/N_1 + 1/N_2\right)^{1/2}}$$

which, of course, assumes that the universes are normal with a common variance and where $n = N_1 + N_2 - 2$ (then the statistic follows the t distribution). On the other hand, when c is small or large, says Duncan(3), say less than 0.2 or greater than 0.8, the degrees of freedom approach those yielded by estimating σ_1' and σ_2' separately, using $N_1 - 1$ and $N_2 - 1$.

Transformation of Variables

It was also decided that in these experiments consideration would be given to the possibility of transforming the original data so that the transformed variables may be more reasonably subjected to the standard forms of analysis.

Formally, we seek to replace our observations X_{ijk} by values of a mathematical function of X_{ijk}, say $f(X_{ijk})$. We would like to select our function $f(X_{ijk})$ so that the new variables $Y_{ijk} = f(X_{ijk})$ are more nearly homoscedastic. Johnson & Leone(7) suggest a method using "statistical differentials" to determine an appropriate form for the function f(x).

1. For example, if the standard deviation of x is proportional to its expected value, then . . . it can be shown that if we let some function say $h(x) = x^2$ we have

$$f(x) \propto \int^{x} (h(x))^{-\frac{1}{2}} dx$$

or substituting

$$f(x) \propto \int_{0}^{x} x^{-1} dx = \ln x$$

Therefore, we consider using the transformation

$$f(x) = \ln x$$

2. As a second example, consider a Poisson type original variable x. In this case the variance is equal or proportional to the expected value, and so h(x) is equal or proportional to x and we have:

$$f(x) \propto \int^{x} x^{-\frac{1}{2}} dx = \sqrt{x}$$

Therefore, we consider using the transformation(7)

$$f(x) = \sqrt{x}$$

These methods of transformation, in an attempt to normalize variances (and another one applicant for the

binomial pattern) are summarized in the table below:

Transformations to Normalize Variances

Name	Relation of Variance to Mean Z	Transformation	Approx. Variance on New Scale	Dist. Inverse Transform
Square Root	var $\propto \bar{z}$	$\sqrt{x}$	.25	Poisson
Nat'l Log	var $\propto \bar{z}^2$	ln x	$(x)^2$	Empirical
Arc Sin	var $\propto \frac{\bar{z}(1-\bar{z})}{N}$	$\sin^{-1}\sqrt{x}$	.25/n	Binomial

For the measure of performance determined by the number of Basic Errors occurring, the Square Root transformation will be used. For the Error Reduction Rate measure of performance (a measure expressed as a percentage between zero and one), the Arc Sin transformation will be used. For the measure of performance determined by the Maximum Error Run Units and the other performance criteria measuring debugging times, the Natural Log transformation will be used.

Section 5

Results from Experiments

The experiments were conducted during August, 1968, for the 16 subject-problem exercises. All of the documentation was collected and appears in the following Appendixes:

1. Sample of documentation from time-sharing terminal session for Invoice Preparation problem - Appendix 1.
2. Sample of documentation from batch computing session including program listing for Inventory Simulation problem - Appendix 1.
3. Set of Program Development-Error Analysis forms for 16 subject-problem exercises including description, classification and existence of all errors - Appendix 2.
4. Set of Performance-Time Record forms for 16 subject-problem exercises showing recapitulation of subject time by work element - Appendix 3.
5. Subject biographical and test score data - Appendix 4.

In order to conduct the analysis of the experimental results, it was necessary to organize the data and calculate the measures of effectiveness for the error analysis. As described earlier, the measurement criteria for each cause and detection level included:

1. total number errors committed (basic)

$$E_{ij} = \sum_{1}^{k} e_{ijk}$$

2. maximum error run units

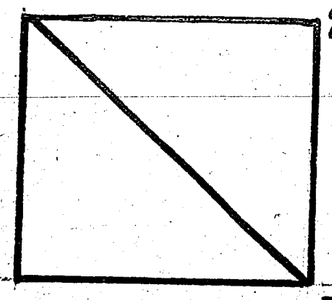
$$R_{ij} = \max \left\{ r_{ijk} \right\}$$

3. error reduction rate

$$ERR_{ij} = \left\{ \frac{\sum_{1}^{k} e_{ijk}}{\sum_{1}^{k} r_{ijk}} \right\}$$

Each of the error criteria were determined and along with the remaining biographical data and other measures of performance, all of the data was summarized on the form Summary Data Sheet. Note that element times are in minutes with debugging time shown for both net and elapsed time. Information appears in the Error Analysis Matrix in the following format:

Error Analysis Matrix



Interpretation

Error No.	Units They Lasted
1	5
2	4
2	9

The set of Summary Data sheets may be found in Appendix 5.

In the conduct of the experiments, the author functioned as an observer in 100% monitoring of the subject's performance. In this capacity the author was able to correct any errors or answer any questions immediately from the subject related to the subject's lack of knowledge or familiarity with the time-sharing system. This was anticipated for the subjects probably were on a learning curve in so far as efficiently using the features of time-sharing and knowing its requirements. This problem of having to learn to efficiently use the features and know the requirements of a new and different system of computing is one that was encountered in most of the previously reported experiments. To the author's knowledge this represents the first attempt to account for this pheonoma in the experimental procedure during the actual experiments. A sample documentation of this monitoring process is shown in Appendix 6.

Section 6

Analysis

To perform an analysis of the results gathered from this experiment and determine how well the objectives have been met we must proceed with the statistical analysis.

Statistical Results

A computer program was written which would accept the experimental data and calculate the appropriate statistics including the following:

1. transformation of data (where appropriate)
2. calculate mean data values for each mode of programming
3. determine Mean Square ANOVA ratio (for F test)
4. calculate Bartlett's M/C to test for validity of variance assumption.

A sample of the output of this program for sample data with suitable explanations appears in Appendix 7. Note that this analysis was performed for all measures of performance where sufficient data was derived from the experiments. A summary of the statistical results is included as follows: Other Performance Criteria (Table 6-3, page 91), Error Reduction Rate (Table 6-4, page 93), Basic Errors (Table 6-5, page 95) and Maximum Error Run Units (Table 6-6, page 97).

The interpretation of these statistical results can be as follows:

1. If Bartlett's M/C Ratio has a value less than the chi-square at the 0.05 level of 7.82, then it is reasonable to conclude that the assumptions required to use ANOVA are upheld.
2. If the Mean Square Ratio has a value greater than that required by the F statistic for some statistical level, it indicates that there is significant difference between the two modes of programming at that level P of confidence.
3. The conversational and batch mean indicates the magnitude and direction of the indicated differences between the two programming modes.

These tables demonstrate how the transformations of raw data can provide greater sensitivity in identifying statistical differences in the results. For example, the Basic Error measure (Table 6-5), the TRANSFORMATION cause errors had a Mean Square Ratio of .9742 in RAW form. When transformed with the Square Root transformation, this same class of errors showed a Mean Square Ratio of 1.2925, significant at the 30% level.

Sample Size and Amount of Data Gathered

The table below shows the number of subject-problem occurrences of at least one measurable error; the limit would be 16 if all subjects had made at least one error of that type in both the problems. (see red boxed cells)

Table 6-7

No. of Data Points Recorded in Experiments by Error Classification

Maximum = 16 No. of subject problems

	DIAGNOSTIC COMPILE DETECTED	DIAGNOSTIC EXECUTION REJECTED	SYSTEM EXPRESSION	LOGIC	LOGICAL INCOMPLETENESS - EXTENDED	CAUSE TOTAL
Misinterpretation	0	3	6	2	7	12
Transformation	12	8	16	12	0	16
Entry	10	8	10	1	0	14
Detection Level Totals	14	12	16	12	7	16 GRAND

Without the existence of at least one error for every subject-problem, the statistical analysis used here is inappropriate. The use of more demanding, complex, time consuming programming tasks would undoubtedly increase the probability of each subject committing at least one of each type error (as defined by the

Cause-Detection Level Matrix). The reader will also note that only four cells had no vacancies, i.e. at least one error occurrence for every subject-problem. As can be seen from Table 6-1, they are:

1. Transformation - System
2. Transformation (All Detection Levels)
3. System (All Causes)
4. Grand (All Causes and Detection Levels)

Note the location of these in the Cause-Detection Level Matrix. The statistical model incorporated in the experimental design requires 16 data points in the 2 x 2 Latin Square Analysis. Using fewer than 16 points in the ANOVA model would involve using one of the methods for handling missing values such as deriving an unbiased linear estimator of the expected value of the missing value. Obviously, this would mean a weaker test with fewer degrees of freedom. It is clear that larger size experiments are needed.

Further Verification of the Classification System

The casual observer might argue that the definitions of error causes and detection levels were poorly designed resulting in an inadequate distribution or differentiation of the errors. To demonstrate the

reasonableness of the error classification scheme examine Table 6-2. This shows the same classification with a total count of the number of errors occurring in each cell of the matrix. Notice the spread in significant number of errors that did occur with this small experiment.

Table 6-2

Total Number of Errors Occurring

	DIAGNOSTIC COMPILE DETECTED	DIAGNOSTIC EXECUTION REJECTED	SYSTEM EXPRESSION	LOGIC	LOGICAL INCOMPLETENESS - EXTENDED	CAUSE TOTALS
Misinterpretation	(0)	5	6	2	12	25
Transformation	24	13	[57]	21	(0)	[115]
Entry	41	10	20	1	0	72
Detection Level Totals	65	28	[83]	24	(12)	[212] GRAND

The three cells with circles indicate no errors occurring and a review of the cause-detection level definitions show clearly that only on rare occasions would we expect errors of this type to occur. For example, an error caused by misinterpretation would not be detected by syntax checking in the FORTRAN compiler.

Mode Differences in Time

The results from the analysis of variance, as shown in Tables 6-3 through 6-6, are not very revealing.

Table 6-3

Statistical Results of Performance Criteria:

Other Performance Criteria

(Square Root Transformed Data)

CRITERIA	CONVERSATIONAL MEAN	BATCH MEAN	MEAN SQUARE ANOVA RATIO	BARTLETTS M/C RATIO
System and Flow chart time	93.50	73.375	.3848	2.4453
Coding Time	187.375	154.125	.7633	.7854
Loading Time	82.25	74.625	.3127	1.1361
Net Debugging Time	180.375	198.375	.0382	6.5982
Elapsed Debugging Time	257.625	750.25	10.1707	5.2262
Program Size (Modified)	110.375	111.75	.0168	5.5745
Total Runs Executed	9.000	7.25	1.1963	3.0625

F statistics for $n_1 = 1, n_2 = 6$

P	f
0.500	.515
0.100	3.78
0.050	5.99
0.025	8.81
0.010	13.7
0.005	18.6

For Bartlett's M/C ratio: $\chi^2_{0.05,n=8} = 7.82$

Differences in any of the measurement criteria are not statistically significant except for Elapsed Debugging Time. This may be due to the fact that:

1. There are no statistical differences between the two modes.
2. The differences are overshadowed by individual differences.
3. The sample size and thus data gathered is small.

Conversational time-sharing is obviously faster than batch when substantial turnaround times are included. From Table 6-3 we have:

Criterion	Conversational Mean	Batch Mean	Mean Square Ratio
Debugging Time-Elapsed	258 Min.	750 Min.	10.17

While this result may be seemingly obvious, the significance of it may not. If, for example, the time spent waiting for a run to be returned under the batch mode is not used productively, then there is a strong advantage for time-sharing in terms of total man-hours involved and elapsed time required to complete a job. If that turnaround period is used completely for other tasks, then it is implied that there is no difference in time consumed for the two methods. This would be in effect Net Debugging Time which shows no real differences.

Table 6-4

Statistical Results of Performance Criteria:

Error Reduction Rate

(Arc Sin Transformed Data)

CAUSE DETECTION LEVEL	CONVERSATIONAL MEAN	BATCH MEAN	MEAN SQUARE ANOVA RATIO	BARTLETTS M/C RATIO
SYSTEM	.2942	.2981	.0022	2.4739
GRAND	.3930	.3568	.1804	4.9948
TRANSFORMATION	.2770	.3786	2.3566	.7728
TRANSFORMATION-SYSTEM	.3736	.3242	.1680	6.1026

F statistics for $n_1 = 1, n_2 = 6$

P	f
0.500	.515
0.100	3.78
0.050	5.99
0.025	8.81
0.010	13.7
0.005	18.6

For Bartlett's M/C ratio: $\chi^2_{0.05,n=3} = 7.82$

Criterion	Conversational Mean	Batch Mean	Mean Square Ratio
Net Debug Time	180 Min.	198 Min.	.038

TRANSFORMATION Errors

To continue with the statistical analysis, we turn our attention to the classification cause TRANSFORMATION. At approximately the .20 significance level, the Error Reduction Rate for conversational time-sharing is lower than batch processing on TRANSFORMATION type errors. (Table 6-4)

Criterion	Conversational Mean	Batch Mean	Mean Square Ratio
ERR Trans-formation	.277	.379	2.36

Thus, at the 20% level we can conclude that errors committed in the process of translating a desired process from a problem description into a set or sequence of programmable steps seem to take more executions or runs to be discovered, detected, and corrected when the task is performed under conversational time-sharing than under batch processing.

We next see that under Basic Errors committed Table 6-5. the TRANSFORMATION cause is again very weakly significant (30% level). Subjects working under conversational

Table 6-5

Statistical Results of Performance Criteria:

Basic Errors

(Square Root Transformed Data)

CAUSE DETECTION LEVEL	CONVERSATIONAL MEAN	BATCH MEAN	MEAN SQUARE ANOVA RATIO	BARTLETTS M/C RATIO
SYSTEM	5.75	4.625	.1583	1.7235
GRAND	14.0	12.375	.2143	.5736
TRANSFORMATION	6.5	7.875	1.2925	1.9206
TRANSFORMATION-SYSTEM	3.50	3.625	1.0812	4.5205

F statistics for $n_1 = 1, n_2 = 6$

P	f
0.500	.515
0.100	3.78
0.050	5.99
0.025	8.81
0.010	13.7
0.005	18.6

For Bartlett's M/C ratio: $\chi^2_{0.05,n=3} = 7.82$

time-sharing, committed about 12% fewer errors of the TRANSFORMATION type than they committed when they were working under the batch operation (Note that overall the greater number of errors were made under conversational time-sharing).

Criterion	Conversational Mean	Batch Mean	Mean Square Ratio
Basic Errors Transformation	6.5	7.88	1.29

Under the third criteria of error measurement, Maximum Error Run Units, we again find a very weak (30% level) significant difference for TRANSFORMATION-cause errors. The maximum error lasted longer in terms of number of executions when using conversational time-sharing than the batch operation. Remember, the life of the error is measured by the number of reruns or executions for which the error exists. This difference was almost two additional executions (average) or an increase of 32% number runs.

Criterion	Conversational Mean	Batch Mean	Mean Square Ratio
MERU Transformation	7.25	5.5	1.80

We also note under the Maximum Error Run Units, Table 6-6, that the Transformation - System cell shows

Table 6-6

Statistical Results of Performance Criteria:

Maximum Error Run Units

(Natural Log Transformed Data)

CAUSE DETECTION LEVEL	CONVERSATIONAL MEAN	BATCH MEAN	MEAN SQUARE ANOVA RATIO	BARTLETTS M/C RATIO
SYSTEM	6.500	5.3750	.2729	2.7704
GRAND	7.2500	6.1250	.2541	3.3385
TRANSFORMATION	7.2500	5.5000	1.7988	1.4735
TRANSFORMATION-SYSTEM	6.25000	4.3750	2.1713	3.1172

F statistics for $n_1 = 1, n_2 = 6$

P	f
0.500	.515
0.100	3.78
0.050	5.99
0.025	8.81
0.010	13.7
0.005	18.6

For Bartlett's M/C ratio: $\chi^2_{0.05,n=3} = 7.82$

a weak significance in the same manner as was just discussed for the Transformation Cause.

Criterion	Conversational Mean	Batch Mean	Mean Square Ratio
MERU Transformation	6.25	4.375	2.17

Effect of Individual Subjects Differences

The large individual differences in the performance of the subjects used in the experiments tends to overshadow the experimental effects of programming mode for most of the error classifications. As an illustration of this, Appendix 7 shows the output of the ANOVA program. The sum of the squares term for subjects is almost twice the term for programming mode (page 187 - Error Reduction Rate). For Maximum Error Run Units (page 190), the difference is more than ten times as much. This should not come as a great surprise since in out critique of other related research every experimenter noted with dismay the tremendous magnitude of individual variance compared to mode variance. As Sackman found, individual variance was in some cases bigger by an order of magnitude. One alternative would be to find a suitable set of determinable factors which would correlate well with mean subject performance. The large effects of skills differences could be partialled out of the analysis, thus possibly leading

to more significant results. In other words, instead of evaluating the variance between observed values and the set mean of these values, one would calculate as the sum of squares the variance between the observed and the predicted:

$$S_i' = \text{partial variance} = \sum \left\{ \text{observed}_i - \text{predicted}_i \right\}^2$$

This results in an analysis of variance using covariance.

Table 6-7 shows data for four independent factors which were collected:

1. Measure of programming training education and experience.
2. Academic performance - grade point standing.
3. Ranking (relative) of subjects performance against rest of subjects of quality and effectiveness of job performance by staff.
4. Scores from test RPAT - Revised Programmer Aptitude Test.

The nine factors shown in Table 6-8 were all possible independent factors and represent the other criteria of performance including Total Errors and Total Runs.

This brings the total number of variables to 13. An intercorrelation matrix was computed as shown in

Table 6-7

Independent Statistical Measures on Subjects

SUBJECT	PGMG. EXP.	ACDM. PERF.	RSCH. RANK	RPAT SCORES	TOTAL SYS & CODE
1	6	3.65	7	44.50	585
2	4	3.48	6	58.25	470
3	1	3.39	1	59.5	740
4	10	3.38	2	60.	300
5	8	3.29	3	67.	624
6	1	3.07	8	62.25	645
7	1	3.48	4	51.	375
8	1	3.27	5	55.75	328
Correlation Variable	1	2	3	4	5

Explanation

Programming Experience - This is a measure of the subjects experience in systems analysis and scientific computer programming. All subjects had approximately one years experience as required in their graduate program. Beyond that approximately one factor credit for every three months experience or 10 hours course instruction.

Academic Performance - This measure is a weighted average of the academic grade point average for a four-year undergraduate curriculum in engineering

and one year of master program in industrial engineering.

Research Ranking - This is a qualitative factor ranking employed by the senior research staff. A rating of 1 is the highest and 8 is the lowest. The rating is supposed to be an evaluation of the subjects capability to perform and potential to succeed.

RPAT Scores - Revised programming aptitude test scores administered to all subjects. This test consists of three sections covering mathematical pattern recognition, spacial relations and geometric reasoning, logic and induction. It is designed to provide an indicator for aptitude in systems and programming skills.

Total System and Coding Time - The total time, in minutes, consumed in the system design flow charting and writing of code by the subject for both experimental problems.

Table 6-8

Additional Criteria - Measures of Performance

PROBLEM	SUBJECT	SYST (MIN)	CODE (MIN)	LOAD (MIN)	DEB NET (MIN)	DEB ELAP (MIN)	PGM EFF	TOTAL ERRORS	TOTAL RUNS
Batch Invoice	5	27	178	57	87	238	105	7	7
	6	70	205	90	628	1238	142	22	14
	8	35	70	45	84	463	64	9	5
	7	60	80	95	145	501	109	10	6
Batch Inventory	4	55	95	75	115	691	131	6	6
	2	65	105	105	151	572	119	15	7
	1	60	275	250	212	718	114	10	5
	3	215	225	80	165	781	110	21	8
Conversation Invoice	1	55	195	110	235	235	121	14	6
	3	45	255	100	269	474	125	23	13
	4	15	135	55	178	228	86	11	5
	2	70	230	73	144	220	111	13	12
Conversation Inventory	8	123	100	69	110	140	105	12	7
	7	105	130	75	65	245	123	7	5
	5	145	274	100	79	79	101	7	6
	6	190	180	76	363	440	111	24	18
Correlation Variables	6		7	8	9	10	11	12	13

INTERCORRELATIVE MATRIX OF INDEPENDENT FACTORS AND ADDITIONAL CRITERIA

	Programming Experience	Academic Performance	Research Ranking	RPAT Scores	Total System & Coding Time	System Time	Coding Time	Loading Time	Debug Net Time	Debug Time Elapsed	Program Efficiency	Total Errors	Total Executed Runs
Programming Experience	1.000												
Academic Performance	.253	1.000											
Research Ranking	-.257	-.089	1.000										
RPAT Scores	.178	-.739	.335	1.000									
Total Systems & Coding Time	-.193	-.137	.045	.233	1.000								
System Time	-.394	-.311	.026	.192	.398	1.000							
Coding Time	.107	.101	.029	.104	.801	.258	1.000						
Loading Time	-.153	.126	.041	-.005	.369	.191	.182	1.000					
Debug Time Net	-.292	-.445	.503	.088	.461	.066	.301	.249	1.000				
Debug Time Elapsed	-.264	-.218	.207	-.011	.281	.041	.070	.057	.683	1.000			
Program Efficiency	-.075	.069	.176	-.069	.359	.104	.284	.564	.510	.470	1.000		
Total Errors	-.551	-.365	.218	.145	.637	.401	.341	.354	.735	.459	.338	1.000	
Total Executed Runs	-.426	-.583	.344	.385	.530	.345	.335	.226	.700	.297	.371	.810	1.000

Table 6-9

Table 6-9 which indicates the correlation between each of these variables. It is evident that none of the collected measures of the subject (Table 6-9) was well enough correlated ($r \geq .5$) to serve as a reliable predictor for mean subject performance in terms of the measurement criteria (Table 6-8), and thus the technique of "partial" analysis of variance was not used.

Summary

From the summary of the analysis of variance results one can now attempt to draw some conclusions. The limited size of the experiment and resulting data gathered from it reduces the degree of statistical discrimination with which the analysis can be made. The following statements can be made at the approximate 80% confidence level. This means that there is about a one in five chance that the difference between the batch and conversational time-sharing modes (for the error measures recorded) was due to chance variation and other "error" in the experiments and that really there was no difference between the modes. The following comments are made with this fact in mind.

1. For the TRANSFORMATION-cause of errors: fewer errors of the cause result under time-sharing than under batch. However, when they are committed, they are harder to find and persist longer in terms of the number of executions required to detect and correct them.

2. From the very concept and design of conversational compilers, ENTRY cause of errors, under time-sharing, are detected sooner (require fewer executions) because most ENTRY errors under time-sharing are detected at the DIAGNOSTIC level (See Table 6-7). This diagnosis is incremental under time-sharing as the text is entered from the terminal before any executions are started.

3. TRANSFORMATION errors, however, are those committed because of mistranslation of the correct mental interpretation into a set of programmable steps. More than half of the errors committed of this type are not detected or diagnosed until the system level TRANSFORMATION-SYSTEM (See Table 6-2, page 90) and only a few (20%) at the Diagnostic-Compiler Detected Level. Imbedded in the system operations and logic of the program as they are; they take longer to find and correct in terms of the number of runs under the conversational time-sharing mode of program preparation.

4. Although MISINTERPRETATION-cause errors did not indicate differences that were statistically significant, the following observation was

made. MISINTERPRETATION errors on the average lasted longer (number of executions) and were harder to detect (logical incompleteness) and fewer of them were committed than errors of the other causes. There were, however, no particular differences apparent between batch and time-sharing for this error cause.

Now recall in Section 4 the proposed definition of the Principle of Forced Interaction:

> An overview of program construction and particularly the relationships of program modules is seriously impaired with the incremental compilation of instant turnaround characteristics of conversational programming on the time-sharing system.

The analysis of these results indicates that for the TRANSFORMATION class of errors by cause, program development under conversational time-sharing results in errors (related to the program construction and logic) which take more effort to detect and correct. This statement is entirely consistent with the Principle of Forced Interaction, and thus supports the assertions of that principle.

Section 7

Conclusions and Recommendations

It was the intent of this experimental research to expand upon the growing body of knowledge related to the interaction of man and machine in the computing environment. In particular, to compare programmer response and performance under the two programming modes - conversational time-sharing and batch operation. The research, to which this experiment was addressed, incorporates several key areas. These areas, such as Error Analysis and Classification and the Principle of Forced Interaction, have not, to the authors knowledge, ever been considered before. The objectives and experimental environment for the research reported herein were formulated from the following experiences:

1. The author's experience in teaching computer programming concepts and scientific language FORTRAN IV.
2. The author's assignment to spend several months evaluating six commercially available time-sharing services (all via TTY-33ASR at Lehigh University).

3. A critical review of the research performed to date in comparing time-sharing and batch systems (this critique was particularly well done by Patrick[10] and Sackman[14]).
4. An investigation into the creation and propogation of entry type errors in a thesis by Smith[19].

Error Classification

It is interesting to note that although little in the way of statistically acceptable results came from the experiment, there was at least indirect affirmation of the plausibility of the error classification scheme. This was in the sense that the errors from the experiments were classified by the error matrix, as originally defined, and at least some errors of almost every type were found. Certainly one measure of a classification scheme such as the one presented herein for programming errors is its stability and consistency. In other words, could the classification be used equally as well for another set of experiments or by another experimenter? This indirectly was accomplished by first applying the scheme to several sample programs under the Process Capability Study.

Principle of Forced Interaction

As shown in the analysis, the experiment did show (weak statistically) the existence of the Principle of Forced Interaction on TRANSFORMATION type errors. TRANSFORMATION errors involved a procedure of operations for performing a desired function correctly interpreted by the programmer, but incorrectly written, as a sequence of programmable steps. These mistakes were detected primarily at the SYSTEM and LOGIC levels (thus no aid from conversational syntax checking) and took more executions or reruns to find and correct under time-sharing. The conclusion is that it takes longer (more runs) because the subject is so close to the segment of the program test in front of him (not the whole listing) and has the very convenient and quick option to reexecute his program rather than look overall for errors (desk checking). This is the assertion made by the Principle of Forced Interaction.

It is interesting to note that ENTRY type errors (although not enough data for a statistical analysis) were not long in life or duration. This seems plausible because of the syntax and diagnostic checking available with both the conversational time-sharing and batch compilers. It has been said that the syntax checking

features of conversational line by line incremental compilers offer real advantage to the programmer. (See Ryan, et.al.[11]). This is demonstrated in two ways:

1. If the programmer makes more than one syntax error in a statement, under a batch system this often means a second or third compile to have the statement completely entry error free.

2. If the programmer is not thorough in checking all the syntax messages, he is prone to further mistakes in making initial corrections, thus requiring additional compiling runs. Under conversational time-sharing, the programmer cannot proceed with the entry (and concurrent compiling) of his program text at the terminal until it is syntax-wise correct.

Recommendations

Thus, the author would recommend that conversational time-sharing systems offer advantages in diagnosing errors in syntax. These are ENTRY type errors primarily. Time-sharing systems have drawbacks, however, in the detection of TRANSFORMATION cause errors where there exists the Principle of Forced Interaction. This recommendation must be tempered with the fact that the experiments conducted (although admittedly small in

scale) were designed to simulate the small engineering research or production shop in industry.

Although this perhaps represents the first work on overall programming error analysis and classification, it should by no means be the last. Sackman set the stage when he said in 1968:

> No work has been done on a comparative error analysis of user performance between time-sharing and batch processing except for some preliminary tabulations listed by Smith(14).

What we have demonstrated in this experiment is the need for continuing creative research and experimentation in the field of man/computer interaction. This means larger experiments, particularly because of the extreme variations in the way seemingly homogenous subjects perform. Perhaps many more statistical conclusions could have been drawn from this experiment had twice as many subjects been employed.

Future experiments could be designed to explore the sequences of responses that the system would provide the user to minimize his learning and error detection functions. Creative experiments are needed on the factor-analytic structure of individual differences in man/computer performance, on systematic changes in user attitudes and

motivation, on the variability and extent of human error, and on the behavioral aspects of problem solving in both closed and open-ended tasks.

It is unfortunate that restrictions on time and money so frequently hamper our quest for more experimentation in this crucial field;for we find today that there is no substantive scientific tradition of applied experimental work dealing with the man/computer interface.

Appendix Listing

Appendix 1

```
// JOB T

LOG DRIVE    CART SPEC    CART AVAIL   PHY DRIVE
  0000         0001         0001         0000

// FOR
*IOCS (CARD,1132PRINTER)
*ONE WORD INTEGERS
* LIST ALL
      INTEGER DEM,SS,OINV,CINV,ORD,REO,DIST,TS,SOH,CYC,TCS,TSOH,TCSS,TSS
     /,P
      DIMENSION XX(20),FREQ(20),CUM(20)
      READ(2,10) ORD,REO
   10 FORMAT(2I5)
      READ(2,20) OCOST,CCOST,SCOST
   20 FORMAT(3F6.2)
      READ(2,30) DIST,NC,LDT,IX
   30 FORMAT(I1,2I3,I5)
      IF(DIST-2) 33,32,33
   32 READ(2,35) EXVA
   35 FORMAT(I5)
      GO TO 40
   33 CUM(I)=0.00
      I=1
   36 READ(2,37) XX(I),FREQ(I)
   37 FORMAT(I5,F5.4)
      NN=I-1
      CUM(I)=CUM(I)+CUM(NN)
      I=I+1
      IF(XX(I)) 36,40,36
   40 KKK=I-1
      CINV=REO
      OINV=REO
      TS=REO
      SOH=REO
      CYC=0
      TCS=0
      TSOH=0
      TCSS=0
      TSS=0
      P=0
   50 IF(TS-REO) 100,100,60
   60 CALL RANDU (IX,IY,YFL)
      IX=IY
      IF(DIST-2) 62,61,62
   61 YY=-(ALOG(YFL)*(EXVA))+095
      DEM=YY
      GO TO 70
   62 DO 70 I=1,KKK
      J=KKK+2-I
      KK=KKK+1-I
      TOP=CUM(J)
      BOTT=CUM(KK)
      IF(TOP-YFL) 64,63,64
   63 DEM=XX(KK)
      GO TO 70
```

This document is the printer listing of the 2nd run showing the errors committed, the subject's recognition of those errors and the pencil notes the subject made to correct them.

Subject: R. Leaser
Program: Inventory Simulation
Date: 8-28-68
Pages: 114-117

```
   64 IF(BOTT-YFL) 66,65,70
   65 DEM=XX(KK)
      GO TO 70
   66 DEM=XX(J)
   70 CONTINUE
      P=P+1
      SOH=CINV
      IF(SOH) 71,72,72
   71 SOH=0
   72 OINV=CINV
      SS=-OINV
      IF(SS) 73,74,74
   73 SS=0
   74 TSS=TSS+SS
      TSOH=TSOH+SOH
      CINV=OINV-DEM
      IF(P-LDT) 75,76,76
   75 TS=TS-DEM
      GO TO 50
   76 CINV=CINV+ORD
      TS=CINV
      GO TO 50
  100 TS=ORD+SOH
      CYC=CYC+1
      P=0
      TCS=TSOH+TCS
      TSOH=0
      TCSS=TSS+TCSS
      TSS=0
      IF(CYC-(NC+1)) 50,101,101
  101 FNC=NC
      COST=OCOST+(CCOST*(TCS/FNC))+(SCOST*(TCSS/FNC))
      WRITE(3,110)
  110 FORMAT(///,1H ' * * * * * * * * * *',' INVENTORY SIMULATION * * *
     / * * * * * * *')
      WRITE(3,111) OCOST,ORD
  111 FORMAT(3X,' ORDERING COST',7X,F6.2,10X,'ORDER QUANTITY',5X,I5)
      WRITE(3,113) SCOST,NC,LDT
  113 FORMAT(3X,'COST UNIT SHORTAGE',2X,F6.2,10X,'NO OF CYCLES',/,42X,'
     /SIMULATED',//,8X,'LEAD TIME IS',I3,' PERIODS')
      IF(DIST-2) 114,125,114
  114 SUM=0.0
      DO 120 I=1,KKK
      SUM=SUM+((CUM(J)-CUM(KK))*XX(KK))
  120 EXVA=SUM
      WRITE(3,121) EXVA
  121 FORMAT(//,27X,'DEMAND HAS EMPIRICAL DISTRIBUTION',/,37X,'WITH MEAN
     /',F6.4,///,23X,'COST AVERAGES PER CYCLE')
      GO TO 130
  125 WRITE(3,126) EXVA
  126 FORMAT(//,27X,'DEMAND HAS EXPONENTIAL DISTRIBUTION',/,37X,'WITH ME
     /AN',F6.4,///,23X,'COST AVERAGES PER CYCLE')
  130 CCC=TCS/FNC
      CSC=TCSS/FNC
      WRITE(3,131) TCS,CSC,OCOST, COST
  131 FORMAT(//,12X,'INVENTORY COST',12X,F6.2,/,17X,'SHORTAGE COST',12X, F6
```

WRITE (3,112) CCOST, REO
112 FORMAT (3X, 'COST CARRY UNIT INV', X, F6.2, 10X,
'REORDER POINT', 6X, I5)

1H1

add * * * ' ///)

2X

', I3, '

ccc

/,22X,'ORDERING COST',12X,F6.2//,27X,'TOTAL COST',15X,F6.2)

```
CORE REQUIREMENTS FOR
 COMMON        0  VARIABLES     176  PROGRAM    886

 END OF COMPILATION

// XEQ


* * * * * * * * * * INVENTORY SIMULATION * * * * * * * * * *
  ORDERING COST          5.00          ORDER QUANTITY        12
 COST UNIT SHORTAGE      1.00         NO OF CYCLES
                                          SIMULATED

     LEAD TIME IS100 PERIODS
 COST UNIT SHORTAGE      4.00         NO OF CYCLES
                                          SIMULATED
     LEAD TIME IS


                  DEMAND HAS EXPONENTIAL DISTRIBUTION
                        WITH MEAN1.0000

              COST AVERAGES PER CYCLE

       INVENTORY COST                    3.00
            SHORTAGE COST                  140.69

                  TOTAL COST                        5.00

       INVENTORY COST                 145.69
            SHORTAGE COST
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

Second line not included

Third line is printing twice

shorten

spacing

wrong by order of magnitude

once only

= ordering cost

→ total cost

RUN #2

This document is the printer display out of the 8th execution. The output is correct for the inventory simulation. This run completed this subjects experiment.

Subject: R. Leaser
Program: Inventory Simulation
Date: 8-29-68

```
* * * * * * * * * * INVENTORY SIMULATION * * * * * * * * * * * * *

ORDERING COST          5.00
COST CARRY UNIT INV    0.04        ORDER QUANTITY        12
COST UNIT SHORTAGE     1.00        REORDER POINT          3
                                   NO OF CYCLES
                                      SIMULATED         100
    LEAD TIME IS  4 PERIODS

                    DEMAND HAS EXPONENTIAL DISTRIBUTION
                         WITH MEAN1.0000

                 COST AVERAGES PER CYCLE

       INVENTORY COST               3.00
          SHORTAGE COST                 1.59
             ORDERING COST                  5.00
                 TOTAL COST                     9.59
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

RUN #8 O.K.!

These document listings from the teletype at a conversational time-sharing session show the subject entering text and encountering syntax diagnostics on the 1st and 2nd sheets. The third, fourth, and fifth show the subject debugging the programs with the aid of diagnostic messages and debug (DISPLAY) commands. Note how the IØ DATA ST. ERROR NO. 20 persists for several program executions. The sixth sheet shows the finished output.

Subject: P. Long
Program: Invoice Preparation
Date: 8-27-68
Pages 118-123

```
PLEASE LØG IN:
PLEASE LØG IN: C6;INDE
PRØJ CØDE: THS
IF YØU ARE HAVING TRØUBLE LØGGING IN,
PLEASE CALL C.S.I.  215-667-1700
.
PLEASE LØG IN: C6;INDE
PRØJ CØDE: THSLØNG

READY  8/27  11:23
-FØR
C.S.I. CØNV. FØRTRAN
+ CØM 1:100

      DIMENSIØN INUM(30),ADDR(30,56),ICNT(30)
    1 FØRMAT(/15(1H*),$CALCULATIØN ØF CUSTØMER INVØICESS,10(1H*))
    2 FØRMAT(10X,$SUBJECT = P.J.LØNGS)
   10 FØRMAT(I5,I1,1X,56A1)
   11 FØRMAT(//,20(1H*),$HELLERTØWN PØWER AND ELECTRIC CØMPANY$,
     10(1H*))
   12 FØRMAT(30X,$CUSTØMER INVØICESS)
   13 FØRMAT(5X,18A1,5X,$RATE  ACCT NØ  UNITS USED  CHARGE$)
   14 FØRMAT (5X,18A1,5X,$----  ----  --  ---------  ------S)
   15 FØRMAT(5X,18A1,7X,I1,4X,I5,5X,I7,2X,'$",F7.2)
   16 FØRMAT (72(1H*))
   17 FØRMAT (&INPUT =SZ)
   18 FØRMAT(5X,$ACCT NØT IN FILE$,7X,$RATE  ACCT NØ  UNITS USED
     CHARGES)
   19 FØRMAT (30X,I1,4X,I5,5X,I7,2X,'$",F7.2)
   20 FØRMAT (I1,9X,I5,8X,I7)
   21 FØRMAT(/,SEND ØF JØBS,//)
      ØPEN (4,/DFILE/,INPUT)
      I=0
      MM=1
      DØ 50 I=1,20
      READ (4,10) INUM(I),ICTL,(ADDR(I,J),J=1,56)
```

```
      ICNT(M)=ICNT(J)
      INUM(J)=ISAVE
      ICNT(J)=ISAV
   70 CONTINUE
      IF(M-IEND)60,80,60
   80 WRITE(1,17)
      READ(0,20)ICODE,IACCT,IHOUR
      IF ICODE.EQ.0 GO TO 999

      IF ICODE.EQ.0 GO TO 999;

      IF (ICODE.EQ.0) GO TO 999
      IF (ICODE-2) 110130,150

      IF (ICODE-2) 110130,150;

      IF (ICODE-2)110,130,150
  110 IF(IHOUE-50)111,111,115
  111 RATE=3.00
      GO TO 500
  115 IF(IHOUR-150)116,116,120
  116 RATE=3.00+(IHOUR-50)*0.045
      GO TO 500

      GO TO)500;

      GO TO 500
  120 IF(IHOUR-250)121,121,125
  121 RATE=7.5+(IHOUR-150)*0.035
      GO TO 500
```

```
 13.      19 FORMAT (30X,I1,4X,I5,5X,I7,2X,"$",F7.2)

+ ED 11

17 FORMAT (&INPUT =$Z)
17 FORMAT (SINPUT =$Z)!
+ LI 31:33

 31.      IF (INUM(M)-INUM(J)) 70,70,65
 32.      65ISAVE=INUM(M)
 33.      ISAVE=ICNT(M)

+ ED 32

65ISAVE=INUM(M)
65< >ISAVE=INUM(M)!
+ LI 32

 32.      65 ISAVE=INUM(M)

+ EX


I9-DATA ST. ERR.! 20.      READ (4,10) INUM(I),ICTL,(ADDR(I,J),J=1,56)

+ LI 16:22

 16.      OPEN (4,/DFILE/,INPUT)
 17.      I=0
 18.      MM=1
 19.      50 I=I+1
 20.      READ (4,10) INUM(I),ICTL,(ADDR(I,J),J=1,56)
 21.      IF(ICTL-1)52,55,52
 22.      52 ICNT(I)=I

+ COM 19

DO 51 I=1,20

!

+ COM 23

51 CONTINUE

!

+ ED 24

55 I=I-1
55 I=20

+ EX


I9-DATA ST. ERR.! 20.      READ (4,10) INUM(I),ICTL,(ADDR(I,J),J=1,56)

+ COM 20.5

DISPLAY INUM
```

change & to $ for input field

must have space separating statement and label

OK

why 8

change this so it reads only 20 records and test

again it didn't work!

```
DISPLAY INUM;

DISPLAY(INUM(I))
!
+ EX

IO-DATA ST. ERR.! 20.       READ (4,10) INUM(I),ICTL,(ADDR(I,J),J=1,56)
+ ED 4

10 FORMAT(I5,I1,1X,56A1)
10 FORMAT(I5,I1,1X,56A1)!
+ EX

    12321
    12345
    20106
    22222
    33015
    34567
    39392
    43234
    45255
    52321
    54321
    55655
    56565
    59000
    66741
    68111
    69696
    75530
    88310
    98765
INPUT =3            12321          856

**********SHELLERTOWN POWER AND ELECTRIC COMPANY**********
                    CUSTOMER INVOICES
      LONES, SAMUAL          RATE  ACCT NO   UNITS USED   CHARGE
      131 MADISON AVE        ----  ----  --  ----------   ------
      NEW LONDON,CONN           3    12321         856   $  30.00
**********************************************************************
INPUT =
STOPPED AT 41.

+ ED 5

11 FORMAT(/,20(1H*),$SHELLERTOWN POWER AND ELECTRIC COMPANY$,10(
1H*))
11 FORMAT(/</////>
+ ED 6

12 FORMAT(30X,$CUSTOMER INVOICES$)
12 FORMAT(</>30X,$CUSTOMER INVOICES$</>
+ ED 16

OPEN (4,/DFILE/,INPUT)
```

```
INPUT =
STOPPED AT 41.

+ DE 20.5


+ EX

INPUT =3          12321          856
```

**********HELLERTOWN POWER AND ELECTRIC COMPANY**********

CUSTOMER INVOICES

	RATE	ACCT NO	UNITS USED	CHARGE
LOPEZ, SAMUAL				
131 MADISON AVE				
NEW LONDON, CONN	3	12321	856	$ 30.00

****

```
INPUT =1          12345          176


EXEC. ERR.! 44.      110 IF(IHOUE-50)111,111,115

+ WHY
```

Should have been "IHOUR" instead of "IHOUE"

VARIABLE HAS NOT BEEN ASSIGNED A VALUE OR STATEMENT LABEL IS MISSING

```
+ WI 44

110 IF(IHOUE-50)111,111,115
110 IF(IHOUR
+ XX

INPUT =1          12345          176
```

**********HELLERTOWN POWER AND ELECTRIC COMPANY**********

CUSTOMER INVOICES

	RATE	ACCT NO	UNITS USED	CHARGE
STONEWALL, M. K.				
84 LOUDER PATH				
BAYSHORE, N.Y.	1	12345	176	$ 8.40

**********HELLERTØWN PØWER AND ELECTRIC CØMPANY**********

CUSTØMER INVØICES

	RATE	ACCT NØ	UNITS USED	CHARGE
MADISON, HENRY	----	---- --	----------	------
2389 15TH ST				
CHICAGO,31 ILL.	1	98765	40	$ 3.00

****

INPUT =3 12321 856

test for mis-match or missing record

**********HELLERTØWN PØWER AND ELECTRIC CØMPANY**********

CUSTØMER INVØICES

	RATE	ACCT NØ	UNITS USED	CHARGE
ACCT NØT IN FILE	3	12321	856	$ 30.00

****

INPUT =1 98765 40

**********HELLERTØWN PØWER AND ELECTRIC CØMPANY**********

CUSTØMER INVØICES

	RATE	ACCT NØ	UNITS USED	CHARGE
MADISON, HENRY	----	---- --	----------	------
2389 15TH ST				
CHICAGO,31 ILL.	1	98765	40	$ 3.00

****

INPUT =
STOPPED AT 41.

O.K.

+
+
-LOG

TIME USED 0:2:09 0:0:00 0:51:54

This last display shows the final output for the customer invoice program including test for mismatch
Subject: P. Long
Program: Invoice Prep.
Date: 8-27-68

Appendix 2

PROGRAM DEVELOPMENT—ERROR ANALYSIS

Page 1 of 1

PROBLEM	DATE	SEQUENCE
Invoice Preparation	8-16-68	1
SUBJECT	**MODE**	**ELAPSED PERIOD**
Tom Long	Batch 1130	8-16 to 8-17

NUMBER	CAUSE AND/OR DETECTION LEVEL	EXPLANATION AND OBSERVATIONS	RUN NUMBER
1	T-DC	Attempt to perform arithmetic operations on a subscript in a statement where the array value appears-- incorrectly formed subscript appears	1
2	E-DE	Entered a statement J J 3 followed by K J 3 which should have been K K 3	2
3	T-DE	Branching as a result of GO TO statement omitted and in wrong location caused execution in a tight loop	2
4	M-LI	Failure to dimension sufficiently large because didn't consider sentinel as one full additional record in maximum file size	3
5	T-S	X type spacing in format too big resulting in misread of account code preventing match	4
6	T-S	In write statement, subject failed to include all of list thinking that he had done so	5
7	M-LI	Subject failed to consider possibility charge in account which does not exist on address file	6
		finish	7

Appendix 3

Page 1 of 1

PERFORMANCE–TIME RECORD

Program	Subject	Mode
Invoice Preparation	Tom Long	Batch 1130

WORK ELEMENT	OBSERVED TIMES START	STOP	ELAPSED	DATE	WORK ELEMENT	OBSERVED TIMES START	STOP	ELAPSED	DATE
1	11:07	11:34	:27	8-16	5	9:47	10:05	:18	8-17
9	11:34	1:10	1:36		10	10:05	10:07	:02	
2	1:11	2:33	1:22		5	10:07	10:30	:23	
9	2:33	2:58	:25		9	10:30	11:15	:45	
2	2:58	3:00	:02		10	11:15	11:18	:03	
8	3:00	3:15	:15		5	11:18	11:45	:27	
2	3:15	4:25	1:10		10	11:45	11:50	:05	
6	8:28	9:25	:57	8-17					
10	9:25	9:27	:02						
5	9:27	9:37	:10						
1	9:37	9:43	:05						
10	9:43	9:47	:04						

WORK ELEMENTS				
	:27	1. System and flow chart		6. Keypunching
	2:58	2. Coding		7. Seeking outside aid
		3. Terminal–loading program		8. FORTRAN language assist or review
		4. Executing and Debugging terminal	2:46	9. Non-experiment time
	1:32	5. List–debugging–batch	:16	10. IBM 1130 Run Time

Appendix 4

Subject Biographical and Test Score Data

Subject	No. in Experiment	Programming Experience	Academic Performance		RPAT Test Scores	Years with Company	Years out of School	Age	Research Ranking Overall Performance
			Undergrad.	Graduate					
Tom Long	5	8	3.09	3.50	67.	11	9	35	3
Joe Alexander	1	6	3.30	4.00	44.50	7	7	28	7
Bob Leaser	3	1	3.12	3.67	59.50	3	3	25	1
David Rockwell	6	1	2.82	3.33	62.25	3	7	28	8
George Boardman	2	4	3.29	3.67	58.25	10	4	38	6
Marty Sowis	7	1	3.16	3.80	51.	10	10	32	4
Lloyd Turner	8	1	3.21	3.33	55.75	12	12	40	5
Pete Long	4	10	2.95	3.80	60.	7	8	30	2

Appendix 5

SUMMARY DATA SHEET

SUBJECT	Tom Long 5	MODE	Batch 1
PROBLEM	Invoice Preparation 1	SEQUENCE	First 1

PROGRAMMING EXPERIENCE	8	SYSTEM TIME	27
ACADEMIC PERFORMANCE	3.29	CODING TIME	178
RESEARCH STAFF RANKING	3	LOADING TIME	57
RPAT SCORES	67	DEBUGGING TIME	87/238
		PROGRAM EFFICIENCY RATING	105
		TOTAL NUMBER ERRORS	7
		TOTAL NO. RUNS EXECUTED	7

** ERROR ANALYSIS **

	DIAGNOSTIC COMPILE	DIAGNOSTIC EXE. REJ.	LOGIC	SYSTEM	LOGIC INCOMPLETE	TOTAL CAUSE
MISINTERPRETATION					2-9 .22 6	2-9 .22 6
TRANSFORMATION	1-1 1. 1	1-2 .5 2		2-9 .22 5		4-12 .33 5
ENTRY		1-2 .5 2				1-2 .5 2
TOTAL DETECTION LEVEL	1-1 1. 1	2-4 1.5 2		2-9 .22 5	2-9 .22 6	7-23 .305 6

SUMMARY DATA SHEET

SUBJECT	Dave Rockwell 6	MODE	Batch 1
PROBLEM	Invoice Calculation 1	SEQUENCE	First 1

PROGRAMMING EXPERIENCE	1	SYSTEM TIME	70
ACADEMIC PERFORMANCE	3.07	CODING TIME	205
RESEARCH STAFF RANKING	8	LOADING TIME	90
RPAT SCORES	62.25	DEBUGGING TIME	628/1238
		PROGRAM EFFICIENCY RATING	142
		TOTAL NUMBER ERRORS	22
		TOTAL NO. RUNS EXECUTED	14

** ERROR ANALYSIS **

	DIAGNOSTIC COMPILE	DIAGNOSTIC EXE. REJ.	LOGIC	SYSTEM	LOGIC INCOMPLETE	TOTAL CAUSE
MISINTERPRETATION			1		2-23 .087 13	2-23 .087 13
TRANSFORMATION	5-7 .714 3		3-21 .143 12	5-20 .25 8		13-48 .27 12
ENTRY	5-6 .832 2	1-4 .25 4		1-11 .096 11		7-21 .33 11
TOTAL DETECTION LEVEL	10-13 .77 3	1-4 .25 4	3-21 .143 12	6-31 .194 11	2-23 .087 13	22-92 .239 13

SUMMARY DATA SHEET

SUBJECT	Lloyd Turner 8	MODE	Batch 1
PROBLEM	Invoice Calculation 1	SEQUENCE	Second 2
PROGRAMMING EXPERIENCE	1	SYSTEM TIME	35
ACADEMIC PERFORMANCE	3.27	CODING TIME	70
RESEARCH STAFF RANKING	5	LOADING TIME	45
RPAT SCORES	55.75	DEBUGGING TIME	84/463
		PROGRAM EFFICIENCY RATING	64
		TOTAL NUMBER ERRORS	9
		TOTAL NO. RUNS EXECUTED	5

** ERROR ANALYSIS **

	DIAGNOSTIC COMPILE	DIAGNOSTIC EXE. REJ.	LOGIC	SYSTEM	LOGIC INCOMPLETE	TOTAL CAUSE
MISINTERPRETATION					2-7 .286 4	2-7 .286 4
TRANSFORMATION	3-3 1. 1			4-10 .40 3		7-13 .54 3
ENTRY						
TOTAL DETECTION LEVEL	3-3 1. 1			4-10 .40 3	2-7 .286 4	9-20 .45 4

SUMMARY DATA SHEET

SUBJECT	Marty Sowis 7	MODE	Batch 1
PROBLEM	Invoice Calculation 1	SEQUENCE	Second 2

PROGRAMMING EXPERIENCE	1	SYSTEM TIME	60
ACADEMIC PERFORMANCE	3.48	CODING TIME	80
RESEARCH STAFF RANKING	4	LOADING TIME	95
RPAT SCORES	51	DEBUGGING TIME	145/501
		PROGRAM EFFICIENCY RATING	109
		TOTAL NUMBER ERRORS	10
		TOTAL NO. RUNS EXECUTED	6

** ERROR ANALYSIS **

	DIAGNOSTIC COMPILE	DIAGNOSTIC EXE. REJ.	LOGIC	SYSTEM	LOGIC INCOMPLETE	TOTAL CAUSE
MISINTERPRETATION					2-9 .222 5	2-9 .222 5
TRANSFORMATION	2-2 1.1	1-3 .33 3	1-3 .33 3	4-16 .25 5		8-24 .33 5
ENTRY						
TOTAL DETECTION LEVEL	2-2 1. 1	1-3 .33 3	1-3 .33 3	4-16 .25 5	2-9 .222 5	10-33 .304 5

SUMMARY DATA SHEET

SUBJECT	Pete Long 4	MODE	Batch 1
PROBLEM	Inventory Simulation 2	SEQUENCE	First 1

PROGRAMMING EXPERIENCE	10	SYSTEM TIME	55
ACADEMIC PERFORMANCE	3.38	CODING TIME	95
RESEARCH STAFF RANKING	2	LOADING TIME	75
RPAT SCORES	60	DEBUGGING TIME	115/691
		PROGRAM EFFICIENCY RATING	131
		TOTAL NUMBER ERRORS	6
		TOTAL NO. RUNS EXECUTED	6

** ERROR ANALYSIS **

	DIAGNOSTIC COMPILE	DIAGNOSTIC EXE. REJ.	LOGIC	SYSTEM	LOGIC INCOMPLETE	TOTAL CAUSE
MISINTERPRETATION				1-5 .20 5		1-5 .20 5
TRANSFORMATION			1-2 .50 2	3-7 .428 3		4-9 .444 3
ENTRY		1-1 1. 1				1-1 1. 1
TOTAL DETECTION LEVEL		1-1 1. 1	1-2 .50 2	4-12 .33 5		6-15 .376 5

SUMMARY DATA SHEET

SUBJECT: George Boardman 2	MODE: Batch 1
PROBLEM: Inventory Simulation 2	SEQUENCE: First 1

PROGRAMMING EXPERIENCE	4	SYSTEM TIME	65
ACADEMIC PERFORMANCE	3.48	CODING TIME	105
RESEARCH STAFF RANKING	6	LOADING TIME	105
RPAT SCORES	58.25	DEBUGGING TIME	151/572
		PROGRAM EFFICIENCY RATING	119
		TOTAL NUMBER ERRORS	15
		TOTAL NO. RUNS EXECUTED	7

** ERROR ANALYSIS **

	DIAGNOSTIC COMPILE	DIAGNOSTIC EXE. REJ.	LOGIC	SYSTEM	LOGIC INCOMPLETE	TOTAL CAUSE
MISINTERPRETATION						
TRANSFORMATION	3-4 / 2 / .75	1-3 / 3 / .33	3-15 / 6 / .20	3-12 / 4 / .25		10-34 / .295 / 6
ENTRY	4-4 / 1 / 1.			1-6 / 6 / .167		5-10 / .50 / 6
TOTAL DETECTION LEVEL	7-8 / .875 / 2	1-3 / .33 / 3	3-15 / .20 / 6	4-18 / .222 / 6		15-44 / .341 / 6

SUMMARY DATA SHEET

SUBJECT	Joe Alexander 1	MODE	Batch 1
PROBLEM	Inventory Simulation 2	SEQUENCE	Second 2

PROGRAMMING EXPERIENCE	6	SYSTEM TIME	60
ACADEMIC PERFORMANCE	3.65	CODING TIME	275
RESEARCH STAFF RANKING	7	LOADING TIME	50
RPAT SCORES	44.50	DEBUGGING TIME	212/718
		PROGRAM EFFICIENCY RATING	114
		TOTAL NUMBER ERRORS	10
		TOTAL NO. RUNS EXECUTED	5

** ERROR ANALYSIS **

	DIAGNOSTIC COMPILE	DIAGNOSTIC EXE. REJ.	LOGIC	SYSTEM	LOGIC INCOMPLETE	TOTAL CAUSE
MISINTERPRETATION			1-3 3 .33	1-3 3 .33	1-3 3 .33	3-9 .33 2
TRANSFORMATION	2-2 1 1.		1-4 4 .25	3-6 2 .50		6-12 .50 4
ENTRY	1-1 1 1.					1-1 1. 1
TOTAL DETECTION LEVEL	3-3 1. 1		2-7 .286 4	4-9 .444 3	1-3 .33 3	10-22 .455 4

SUMMARY DATA SHEET

SUBJECT: Bob Leaser 3	MODE: Batch 1
PROBLEM: Inventory Simulation 2	SEQUENCE: Second 2

PROGRAMMING EXPERIENCE	1	SYSTEM TIME	215
ACADEMIC PERFORMANCE	3.39	CODING TIME	225
RESEARCH STAFF RANKING	1	LOADING TIME	80
RPAT SCORES	59.5	DEBUGGING TIME	165/781
		PROGRAM EFFICIENCY RATING	110
		TOTAL NUMBER ERRORS	21
		TOTAL NO. RUNS EXECUTED	8

** ERROR ANALYSIS **

	DIAGNOSTIC COMPILE	DIAGNOSTIC EXE. REJ.	LOGIC	SYSTEM	LOGIC INCOMPLETE	TOTAL CAUSE
MISINTERPRETATION				1-4 .25 4		1-4 .25 4
TRANSFORMATION	3-3 1. 1		3-15 .20 6	5-17 .294 5		11-35 .314 6
ENTRY	6-6 1. 1			3-7 .428 4		9-13 .69 4
TOTAL DETECTION LEVEL	9-9 1. 1		3-15 .20 6	9-28 .322 5		20-52 .385 6

SUMMARY DATA SHEET

SUBJECT		MODE	
Joe Alexander	1	Conversational	2
PROBLEM		**SEQUENCE**	
Invoice Preparation	1	First	1

PROGRAMMING EXPERIENCE	6	SYSTEM TIME	55
ACADEMIC PERFORMANCE	3.65	CODING TIME	195
RESEARCH STAFF RANKING	7	LOADING TIME	110
RPAT SCORES	44.5	DEBUGGING TIME	235/235
		PROGRAM EFFICIENCY RATING	121
		TOTAL NUMBER ERRORS	14
		TOTAL NO. RUNS EXECUTED	6

** ERROR ANALYSIS **

	DIAGNOSTIC COMPILE	DIAGNOSTIC EXE. REJ.	LOGIC	SYSTEM	LOGIC INCOMPLETE	TOTAL CAUSE
MISINTERPRETATION		2-2 1. 1	1-2 .5 2		2-5 .4 4	5-9 .555 4
TRANSFORMATION		1-1 1. 1	2-5 .4 4	2-6 .33 3		5-12 .418 4
ENTRY	5-5 1. 1					5-5 1. 1
TOTAL DETECTION LEVEL	5-5 1. 1	3-3 1. 1	3-7 .428 4	2-6 .33 3	2-5 .4 4	15-26 .577 4

SUMMARY DATA SHEET

SUBJECT	Bob Leaser 3	MODE	Conversational 2
PROBLEM	Invoice Preparation 1	SEQUENCE	First 1
PROGRAMMING EXPERIENCE	1	SYSTEM TIME	45
ACADEMIC PERFORMANCE	3.39	CODING TIME	255
RESEARCH STAFF RANKING	1	LOADING TIME	100
RPAT SCORES	59.5	DEBUGGING TIME	269/474
		PROGRAM EFFICIENCY RATING	125
		TOTAL NUMBER ERRORS	23
		TOTAL NO. RUNS EXECUTED	13

** ERROR ANALYSIS **

	DIAGNOSTIC COMPILE	DIAGNOSTIC EXE. REJ.	LOGIC	SYSTEM	LOGIC INCOMPLETE	TOTAL CAUSE
MISINTERPRETATION		2-3 2 .67		1-10 10 .10	1-11 11 .09	4-24 .167 11
TRANSFORMATION	1-1 1 1.	2-8 5 .25	1-11 11 .09	4-34 9 .12		8-54 .148 11
ENTRY	7-7 1 1.	1-3 3 .33	1-1 1 1.	2-10 9 .2		11-21 .525 9
TOTAL DETECTION LEVEL	8-8 1. 1	5-14 .357 5	2-12 .167 11	7-54 .13 10	1-11 .09 11	23-99 .232 11

SUMMARY DATA SHEET

SUBJECT		MODE	
Pete Long	4	Conversational	2
PROBLEM		**SEQUENCE**	
Invoice Calculation	1	Second	2

PROGRAMMING EXPERIENCE	10	SYSTEM TIME	15
ACADEMIC PERFORMANCE	3.38	CODING TIME	135
RESEARCH STAFF RANKING	2	LOADING TIME	55
RPAT SCORES	60	DEBUGGING TIME	178/228
		PROGRAM EFFICIENCY RATING	86
		TOTAL NUMBER ERRORS	11
		TOTAL NO. RUNS EXECUTED	5

** ERROR ANALYSIS **

	DIAGNOSTIC COMPILE	DIAGNOSTIC EXE. REJ.	LOGIC	SYSTEM	LOGIC INCOMPLETE	TOTAL CAUSE
MISINTERPRETATION						
TRANSFORMATION	1-1 1 1.		1-4 4 .25	2-4 2 .5		4-9 .444 4
ENTRY	5-5 1 1.	1-1 1 1.		1-3 3 .33		7-9 .78 3
TOTAL DETECTION LEVEL	6-6 1. 1	1-1 1. 1	1-4 .25 4	3-7 .428 3		11-18 .611 4

SUMMARY DATA SHEET

SUBJECT: George Boardman 2	MODE: Conversational 2
PROBLEM: Invoice Calculation 1	SEQUENCE: Second 2

PROGRAMMING EXPERIENCE	4	SYSTEM TIME	70
ACADEMIC PERFORMANCE	3.48	CODING TIME	230
RESEARCH STAFF RANKING	6	LOADING TIME	73
RPAT SCORES	58.25	DEBUGGING TIME	144/220
		PROGRAM EFFICIENCY RATING	111
		TOTAL NUMBER ERRORS	13
		TOTAL NO. RUNS EXECUTED	12

** ERROR ANALYSIS **

	DIAGNOSTIC COMPILE	DIAGNOSTIC EXE. REJ.	LOGIC	SYSTEM	LOGIC INCOMPLETE	TOTAL CAUSE
MISINTERPRETATION		1-1 1 1.				1-1 1. 1
TRANSFORMATION	1-1 1 1.	2-9 5 .222	2-17 10 .12	2-20 10 .10		7-47 .149 10
ENTRY	2-2 1 1.			3-10 8 .30		5-12 .417 8
TOTAL DETECTION LEVEL	3-3 1. 1	3-10 .30 5	2-17 .117 10	5-30 .167 10		13-60 .217 10

SUMMARY DATA SHEET

SUBJECT: Lloyd Turner 8	MODE: Conversational 2
PROBLEM: Inventory Simulation 2	SEQUENCE: First 1

PROGRAMMING EXPERIENCE	1	SYSTEM TIME	123
ACADEMIC PERFORMANCE	3.27	CODING TIME	100
RESEARCH STAFF RANKING	5	LOADING TIME	69
RPAT SCORES	55.75	DEBUGGING TIME	110/140
		PROGRAM EFFICIENCY RATING	105
		TOTAL NUMBER ERRORS	12
		TOTAL NO. RUNS EXECUTED	7

** ERROR ANALYSIS **

	DIAGNOSTIC COMPILE	DIAGNOSTIC EXE. REJ.	LOGIC	SYSTEM	LOGIC INCOMPLETE	TOTAL CAUSE
MISINTERPRETATION						
TRANSFORMATION	1-1 1 1.	3-7 3 .428		4-21 6 .19		8-29 .276 6
ENTRY	2-2 1 1.			2-9 5 .222		4-11 .364 5
TOTAL DETECTION LEVEL	1-1 1 1. 1	5-9 3 .555 3		6-30 6 .20 6		12-40 .30 6

SUMMARY DATA SHEET

SUBJECT: Marty Sowis 7	MODE: Conversational 2
PROBLEM: Inventory Simulation 2	SEQUENCE: First 1

PROGRAMMING EXPERIENCE	1	SYSTEM TIME	105
ACADEMIC PERFORMANCE	3.48	CODING TIME	130
RESEARCH STAFF RANKING	4	LOADING TIME	75
RPAT SCORES	51	DEBUGGING TIME	65/245
		PROGRAM EFFICIENCY RATING	123
		TOTAL NUMBER ERRORS	7
		TOTAL NO. RUNS EXECUTED	5

** ERROR ANALYSIS **

	DIAGNOSTIC COMPILE	DIAGNOSTIC EXE. REJ.	LOGIC	SYSTEM	LOGIC INCOMPLETE	TOTAL CAUSE
MISINTERPRETATION						
TRANSFORMATION	1-1 1 1.			2-8 4 .25		3-9 .33 4
ENTRY	3-3 1 1. 4-4			1-1 1 1. 3-9		4-4 1. 7-13
TOTAL DETECTION LEVEL	1. 1			.33 4		.540 4

SUMMARY DATA SHEET

SUBJECT: Tom Long 5	MODE: Conversational 2
PROBLEM: Inventory Simulation 2	SEQUENCE: Second 2

PROGRAMMING EXPERIENCE	8	SYSTEM TIME	145
ACADEMIC PERFORMANCE	3.29	CODING TIME	274
RESEARCH STAFF RANKING	3	LOADING TIME	100
RPAT SCORES	67	DEBUGGING TIME	79/79
		PROGRAM EFFICIENCY RATING	101
		TOTAL NUMBER ERRORS	7
		TOTAL NO. RUNS EXECUTED	6

** ERROR ANALYSIS **

	DIAGNOSTIC COMPILE	DIAGNOSTIC EXE. REJ.	LOGIC	SYSTEM	LOGIC INCOMPLETE	TOTAL CAUSE
MISINTERPRETATION				1-2 .5 2		1-2 .5 2
TRANSFORMATION			1-5 .2 5	2.4 .5 2		3-9 .33 5
ENTRY		101 1. 1		2-2 1. 1		3-3 1. 1
TOTAL DETECTION LEVEL		1-1 11. 1	1-5 .2 5	5-8 .625 2		7-14 .50 5

SUMMARY DATA SHEET

SUBJECT: Dave Rockwell 6	MODE: Conversational 2
PROBLEM: Inventory Simulation 2	SEQUENCE: Second 2

PROGRAMMING EXPERIENCE	1	SYSTEM TIME	190
ACADEMIC PERFORMANCE	3.07	CODING TIME	180
RESEARCH STAFF RANKING	8	LOADING TIME	76
RPAT SCORES	62.25	DEBUGGING TIME	363/440
		PROGRAM EFFICIENCY RATING	111
		TOTAL NUMBER ERRORS	24
		TOTAL NO. RUNS EXECUTED	18

** ERROR ANALYSIS **

	DIAGNOSTIC COMPILE	DIAGNOSTIC EXE. REJ.	LOGIC	SYSTEM	LOGIC INCOMPLETE	TOTAL CAUSE
MISINTERPRETATION				1-11 11 .096		1-11 .096 11
TRANSFORMATION		2-6 4 .33	2-12 10 .167	10-104 14 .096		14-122 .115 14
ENTRY	3-3 1 1.	2-4 3 .50		4-4 1 1.		9-11 .82 3
TOTAL DETECTION LEVEL	3-3 1. 1	4-10 .40 4	2-12 .167 10	15-109 .138 14		24-144 .167 14

Appendix 6

Documented Teletype Assistance

No.	Question	Answer
1.	Do I dimension what has been specified in an integer statement as an array?	Yes
2.	I made a mistake and (CR), how do I correct?	Change Later
3.	Is a dash a minus sign?	Yes
4.	Diagnostic error (hit!) gave <u>program may require an end card</u> - What do I do, just retype it?	Yes
5.	(skipped four statements) - Can I delete last two statements?	Retype from beginning of skipped and later delete 2 redundant statements
6.	Go over sign off procedure (for lunch)	END! SAVE -LOG
7.	How do I recompile and continue to add test?	Load file in short sequence, compile in TTY after file sequence

PROGRAM PRINTOUT SHOWING RESULTS OF ANALYSIS OF VARIANCE:
ANALYSIS OF VARIANCE IN LATIN SQUARE MODEL

RESULTS FOR
ERROR REDUCTION RATE (TRANSFORMATION)

ARSIN SQRT TRANSFORMATION

SOURCE	DEGREES OF FREEDOM	SUM OF SQUARES	MEAN SQUARE	MEAN SQR RATIO
PROGRAMMING MODE	1	.054394029	.054394029	2.356557838
PROBLEM	1	.001929474	.001929474	.083592197
INTERACTION + BLOCKS (SUBJECTS)	1	.089099064	.012728438	.551444715
RESIDUAL	6	.138491900	.023081983	
TOTAL	15	.283914467		

F statistic for $n_1=1, n_2=6$

P	f
0.500	.515
0.100	3.78
0.050	5.99
0.025	8.81
0.010	13.7

ASSOCIATED STATISTICS FOR SAME RUN:
RESULTS FOR

ERROR REDUCTION RATE (TRANSFORMATION) ARSIN SQRT TRANSFORMATION

REPLICATE STATISTICS

ENTRY	NO OF DATA VALUES	MEAN	DEVIATION
BATCH INVOICE PREP	4	.650523450	.052406854
BATCH INVENTORY SIM	4	.670916625	.044485420
CONVERSATIONAL INVOICE PREP	4	.555873700	.080274439
CONVERSATIONAL INVENTORY SIM	4	.532341100	.055266647

VARIANCE FOR LEVELS OF MODE
VAR 1 2.6750690
VAR 2 1.8157157

*the difference between these is a measure of the lack of homoscedasticity

SUM OF(N(I) TIMES LOG(S(I) SQUARED) = -29.912122673
12 TIMES LOG(SUM OF VARIANCES -/- 4.) = -29.388525182
BARTLETTS TEST FOR HOMOGENEITY OF VARIANCES
RATIO M/C = .930992728
CHI SQUARE STATISTIC FOR .05 LEVEL WITH 3 DEG FREE = 7.82

*Bartletts test ratio M/C indicates that the observed differences in variance here are not significant enough to void the use of analysis of variance.

ANALYSIS OF VARIANCE IN LATIN SQUARE MODEL

RESULTS FOR MAX ERROR RUN UNITS (SYSTEM) RAW (NOT TRANSFORMED)

SOURCE	DEGREES OF FREEDOM	SUM OF SQUARES	MEAN SQUARE	MEAN SQR RATIO
PROGRAMMING MODE	1	5.062500000	5.062500000	.268582481
PROBLEM	1	1.562500000	1.562500000	.082895828
INTERACTION + BLOCKS (SUBJECTS)	1	61.218750000	8.745535714	.463979789
RESIDUAL	6	113.093750000	18.848958333	
TOTAL	15	180.937500000		

F statistic for $n_1=1, n_2=6$

P	f
0.500	.515
0.100	3.78
0.050	5.99
0.025	8.81
0.010	13.7
0.005	18.6

RESULTS FOR
MAX ERROR RUN UNITS (SYSTEM)

RAW (NOT TRANSFORMED)

REPLICATE STATISTICS

ENTRY	NO OF DATA VALUES	MEAN	DEVIATION
BATCH INVOICE PREP	4	6.000000000	1.500000000
BATCH INVENTORY SIM	4	4.750000000	.544862368
CONVERSATIONAL INVOICE PREP	4	6.500000000	1.750000000
CONVERSATIONAL INVENTORY SIM	4	6.500000000	2.277608395

VARIANCE FOR LEVELS OF MODE
VAR 1 177.6406250
VAR 2 260.8437500

*Notice in this case conversational mode clearly dominates the batch mode

SUM OF(N(I) TIMES LOG(S(I) SQUARED) = 3.077371057
12 TIMES LOG(SUM OF VARIANCES -/- 4.) = 5.174856985
BARTLETTS TEST FOR HOMOGENEITY OF VARIANCES
RATIO M/C = 3.729475752
CHI SQUARE STATISTIC FOR .05 LEVEL WITH 3 DEG FREE = 7.82

Appendix 8

Test Problem #1

Data Processing Program for Calculation of Customer Invoices

The Hellertown Power and Electric Company has recently acquired a computer to replace some of its tabulating equipment. The electric company has now collected its monthly meter readings and punched them on cards in the format shown below, sorted into ascending order by Customer Account Number.

Columns	Contents	Range of Values
1	Rate Schedule Code	1-3
11-15	Customer Account Number	10000-99999
24-30	Kilowatt Hours Used	0 to 1000000 (in whole numbers)

Write a program that will first read a card, one by one, and calculate the customers electric charge by referring to an appropriate subroutine:

Schedule Code	Range	Rate Charged
1 (Normal house power)	50 kwh or less	$ 3.00 (minimum bill)
	51-150 kwh	$ 3.00 plus $.045 per kwh over 50
	151-250 kwh	$ 7.50 plus $.035 per kwh over 150
	over 250 kwh	$11.00 plus $.03 per kwh over 250
2 (House plus hot water and/or heat)	150 kwh or less	$ 7.50 (minimum bill)
	151-350 kwh	$ 7.50 plus $.03 per kwh over 150
	over 350 kwh	$13.50 plus $.025 per kwh over 350

Schedule Code	Range	Rate Charged
3 (Industrial rate)	1000 kwh or less	$ 30.00 (minimum bill)
	1001-5000 kwh	$ 30.00 plus $0.02 per kwh over 1000
	5001-20000 kwh	$110.00 plus $0.015 per kwh over 5000
	over 20000 kwh	$335.00 plus $0.01 per kwh over 20000

Once the customer's charge is calculated, then look up the customer's address in the address file. This file is available in card image form (no more than 20 records). The file is not presently in ascending customer account number sequence (a sentinel 1 in column 6 indicates end of file).

Customer File Format

Column	Contents
1-5	Customer Account Number
8-25	Customer Name
27-44	Street Address
46-63	City, State, and Zip Code

The last operation would be to print a listing of the invoices for the customers on the printer in the format indicated below. Also, skip to a new page for each new customer bill.

Note: The program should terminate by a sentinel from a card following the last data card (for example "0" in the rate schedule code).

Output Format

* * * * * HELLERTOWN POWER AND ELECTRIC COMPANY * * * * *

CUSTOMER INVOICE

NAME STREET ADDRESS CITY, STATE, ZIP CODE	RATE	ACCT NO	UNITS USED	CHARGE
	x	xxxxx	xxxxxx	$xxx.xx

* *

Test Problem #2

Evaluation of Inventory Control Policy

with Monte-Carlo Simulation

In the analysis of an inventory situation, the objective is frequently to minimize the total inventory and associated replenishment costs (and shortage costs) by choosing in an optimal fashion a fixed order quantity to be ordered each time the level of inventory on hand reaches a certain predetermined level, i.e. the reorder point. The purpose of this program is to simulate an inventory situation where the two policy parameters have been chosen, namely the order quantity and the reorder point. The total cost for inventory storage, shortage, and ordering can be determined from the following formula:

$$\text{Total Cost (per order cycle)} = 1 \times \text{Ordering Cost} + \left\{\begin{array}{l}\text{Cost of carrying one unit in inventory for one period}\end{array}\right\} \times \left\{\begin{array}{l}\text{Total no. of unit periods of stock on hand for cycle}\end{array}\right\} + \left\{\begin{array}{l}\text{Cost of incurring one unit stock of shortage for one period}\end{array}\right\} \times \left\{\begin{array}{l}\text{Total no. of unit periods of shortage for cycle}\end{array}\right\}$$

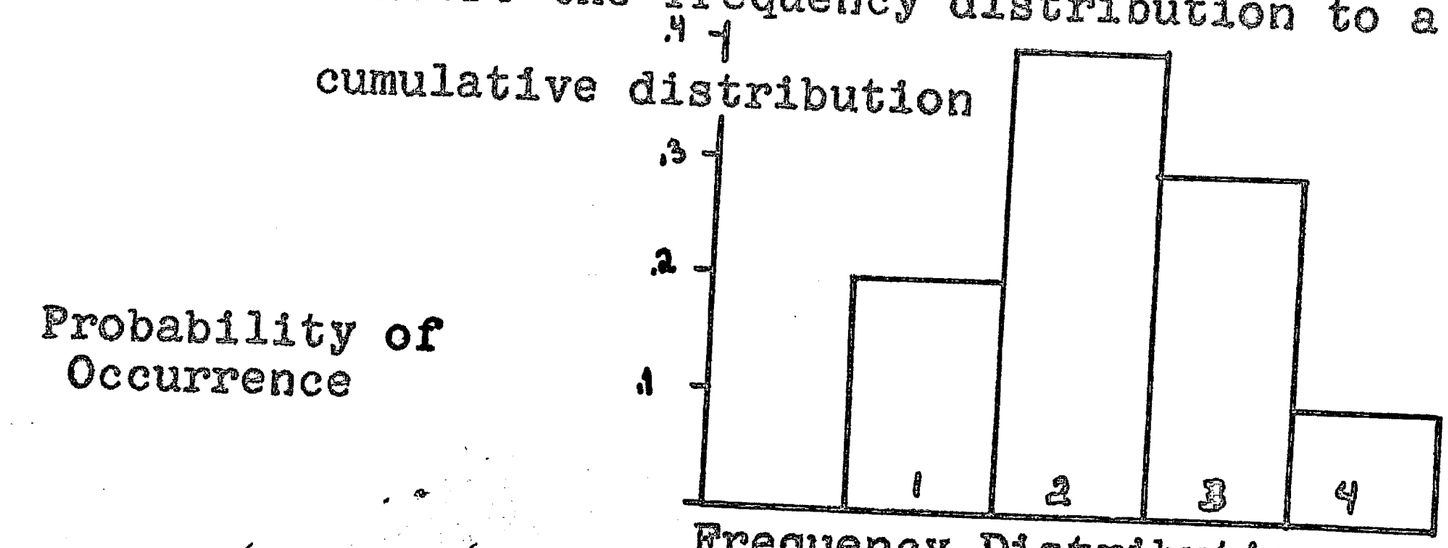
In order to determine the demand for any period, the demand distribution must be sampled in a Monte

Carlo fashion to illustrate. When the level of the stock on hand reaches the reorder point, an order is placed. The demand distribution will be either exponential or empirical in form.

Monte Carlo Sampling of a Distribution

Empirical distribution given a table of intervals and frequencies

1. Convert the frequency distribution to a cumulative distribution



2. With a pseudo-random number generator, generate a random number from a univorm distribution whose range is $0 \leq x_i \leq 1.0$

3. Enter the cumulative distribution with the random number as the probability of occurrence and read the corresponding value.

4. The abscissa value is then the sample value desired.

Note: When abscissa repeated many times, the distribution of the sample values generated will approximate the original frequency distribution.

Exponential distribution given the mean or expected value

The procedure is essentially the same except a table look-up of the values is unnecessary because this distribution can be expressed in an analytical form:

density $f(x) = \alpha e^{-\alpha x}$

cumulative $F(x) = 1 - \alpha e^{-\alpha x}$

expected value $EX = \int_0^\infty x \alpha e^{-\alpha x} dx = \frac{1}{\alpha}$

or; $\alpha = \frac{1}{EX}$

now if 'r' is a pseudo-random number, and because of the symmetry of the uniform distribution from which it came, F(X) and 1-F(X) are interchangeable, therefore:

$r = e^{-\alpha x}$ or $\log r = -\alpha x$

and consequently:

$$x = -\left(\frac{1}{\alpha}\right) \log r = -EX \log r$$

Thus for each value of the pseudo-random number 'r', a unique value of x is determined, which will take on only non-negative values (since $\log r \leq 0$ for $0 \leq r \leq 1$) and will follow the exponential density function with the expected value EX.

The following is a typical cycle of the logical operations in simulating an inventory situation:

1. Begin by setting the stock-on-hand at the reorder point level.
2. Check the stock-on-hand plus any stock-on-order against the reorder point.
3. If the stock-on-(sum) is equal to or less than the reorder point, place an order for the order quantity (draw a sample value from the lead time distribution to determine the number of time periods from now that the order will arrive). Go to 4.
4. If the stock-on-(sum) is greater than (has not reached) the reorder point, proceed to draw a sample demand from

the demand distribution: (or if 3 is completed).

5. This demand value is subtracted from the stock-on-hand, resulting in a new stock level at the end of the period.
6. Now since one period has passed, 1 is subtracted from all outstanding lead time values.
7. If a lead time value is reduced to zero, an amount equal to the order quantity is added to the stock-on-hand.
8. Then the statistical data for the period is calculated and the next period is considered.
9. If a lead time value is not reduced to zero, statistical data for the period is calculated and the next period is considered. Statistical data for the period should include:

a. add one to the period count and record
b. record the units on hand (inventory)
c. or the units short (shortage)

Input Format

Data and parameter values should be read from the card as follows:

Card Number	Columns	Contents
1	1-5	order quantity (units)
	6-10	reorder point (units)
2	1-6	cost of placing an order ($xx.xx)
	7-12	cost of carrying one unit of stock for one period ($xx.xx)
	13-18	cost of one unit short for one period ($xx.xx)
3	1	demand distribution sentinel 1 - empirical 2 - exponential
	2-4	number of cycles (orders) to be simulated
	5-7	lead time (no. of time periods)
	8-12	starting value for random number generator
		Demand Distribution
4 or 4	1-5	expected value number of demand level, frequency pairs
5	1-5,6-10	demand level, frequency pairs
		Demand Frequency
xxxxx	.xxxx	Note: maximum number of pairs is 20

Output Format

* * * * * * * * * INVENTORY SIMULATION * * * * * * * * *

ORDERING COST	x.xx	ORDER QUANTITY	xxx
COST CARRY UNIT INV	x.xx	REORDER POINT	xx
COST UNIT SHORTAGE	x.xx	NO. OF CYCLES SIMULATED	xxx

LEAD TIME IS x PERIODS

DEMAND HAS EXPONENTIAL DISTRIBUTION
WITH MEAN x.xx

COST AVERAGES PER CYCLE

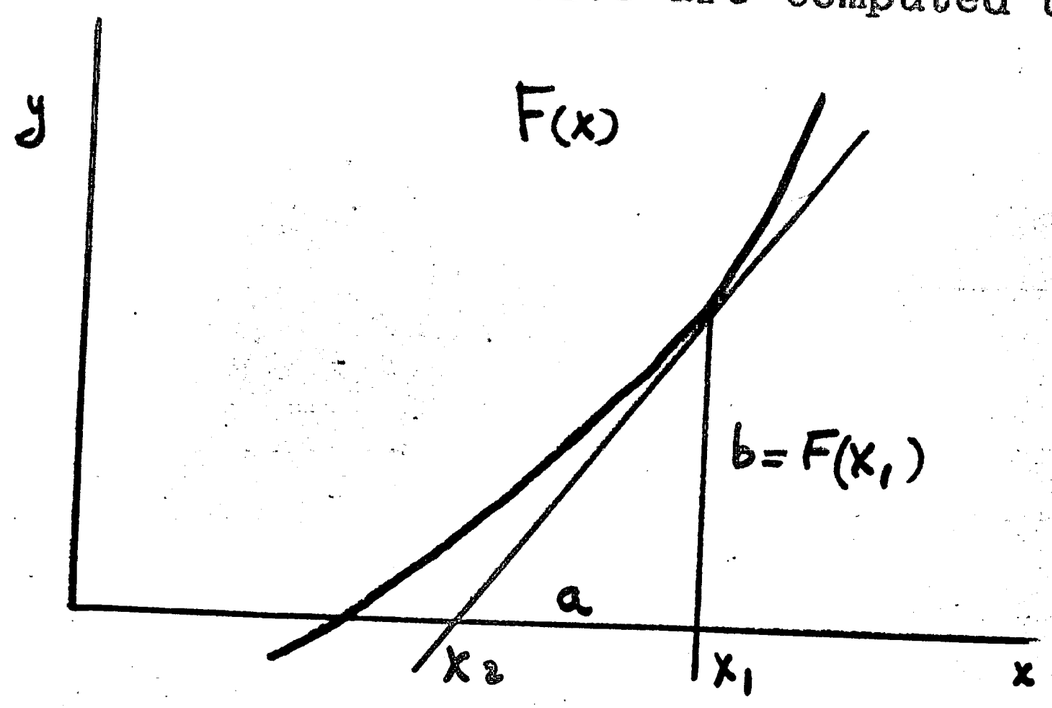
INVENTORY COST	xxx.xx
SHORTAGE COST	xxx.xx
ORDERING COST	xxx.xx
TOTAL COST	xxx.xx

* *

Test Problem #3

Finding the Root of a Polynomial with the Newton-Raphson Method

The Newton-Raphson method for finding a root of an equation is one of the most commonly used numerical methods, being valuable with or without a computer. The method requires that an initial guess of the root be provided. The value of the polynomial and the value of its derivative are computed using this value.



In this figure 'b' is the value of the polynomial at 'x_1' and the slope of the line tangent to the curve is the value of the derivative of the polynomial at 'x_1'.

Hence $\frac{b}{a} = \frac{F(x_1)}{a} = F'(x_1)$ (value of the derivative at x_1)

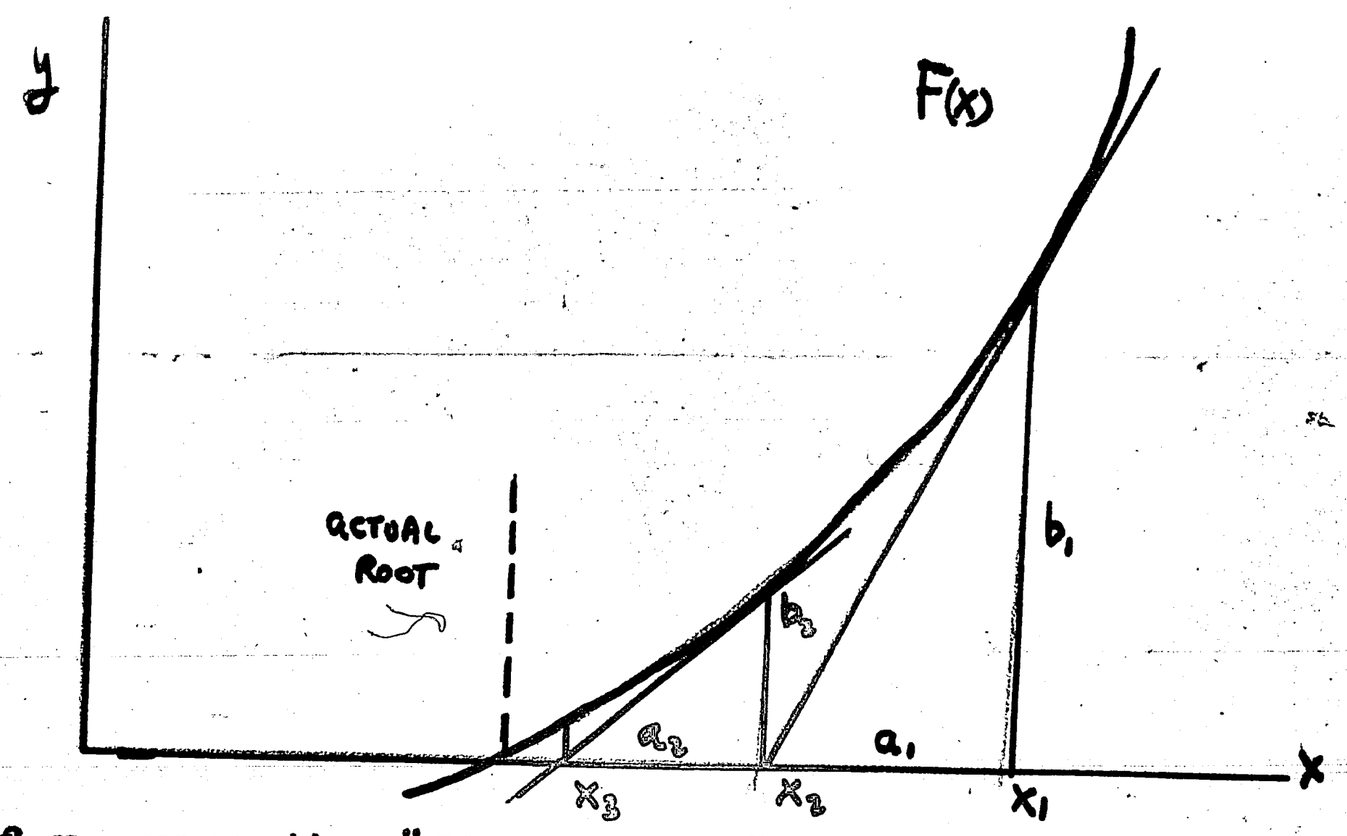
and now rearranging terms: $a = \frac{F(x_1)}{F'(x_1)}$ and $x_1 = x_2 + a$

so that $x_2 = x_1 - a = x_1 - \frac{F(x_1)}{F'(x_1)}$

Thus, the Newton-Raphson method says that given a function of x, $F(x) = 0$, if x_1 is an approximation to the root. A better approximation is given by:

$$x_{i+1} = x_i - \frac{F(x_i)}{F'(x_i)}$$

This formula is solved a sufficient number of times to "converge" on a solution. Each computed x value is substituted back into the right-hand side of the above formula to produce a new x value. This iterating process is demonstrated below:



If x_1 were the "first guess" of the actual root value, the first computation of the formula will produce x_2, repeating using x_2 the formula computes x_3. This process repeats until the latest x is within a prescribed allowable "error amount" of the actual root.

This iteration method, under certain conditions, may be unstable, that is, it might proceed away from the root (diverge) or not converge at all (oscillate). A limit to the number of iterations should be incorporated and an appropriate message if this condition has been reached.

To compute the derivative of the polynomial for use in the correction term $F(x)/F'(x)$ in a subroutine, consider as n^{th} degree polynomial in general form as:

$$a_1 + a_2x + a_3x^2 \quad \ldots\ldots\ldots + a_{n\ 1}x^n$$

The derivative in general is:

$$(a_2) + (2a_3)x + (3a_4)x^2 \quad \ldots\ldots + (na_{n\ 1})x^{n-1}$$

The coefficients in the parenthesis will have to be computed.

When the absolute value of the correction term is less than some error limit, epsilon, then the last computer x is the root approximation good to sufficient accuracy.

Input Format

Card	Contents
1	Read an integer (max 10) describing how many sets of data follow
2	Read an integer as an index limit, then read up to 10 real numbers, the amount to be determined by the integer index limit Note: coefficients are read in order as $a_1, a_2 \ldots$
3	Read a single real number to be used as a starting point (first guess)
4	Read an epsilon (error limit) test value (like 10^{-6})

Output Format

*** * * NEWTON-RAPHSON METHOD OF FINDING POLYNOMIAL ROOT * * ***

DATA SET	DEGREE OF POLYNOMIAL	FIRST GUESS	ROOT APPROXIMATION	POLYNOMIAL COEFFICIENTS
xx	xx	xx.xxxx	xx.xxxx	xx.xxxx
.				
.				
.				

* *

Test Problem #4

Solving a System of Simultaneous Equations by the Gauss-Siedel Method

The Gauss-Siedel method for solving a system of simultaneous equations does not apply for all systems of equations. When it does apply, it is quite attractive, but the method does require the following condition:

> In every row, the main diagonal term (the one with the same row and column number) must dominate the other terms in that row. This means that the absolute value of the main diagonal term must be greater than the sum of the absolute values of all the other terms in the row.

Method of Gauss-Siedel Iteration

Let us illustrate the method in terms of a system of three equations in three unknowns.

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3$$

Suppose we make guesses at the values of x_2 and x_3 - it doesn't matter whether they are good guesses; zeros will work. We solve the first equation for x_1, and writing a prime to indicate that it is a new approximation:

$$x_1' = \frac{b_1 - a_{12}x_2 + a_{13}x_3}{a_{11}}$$

Now using the value for x_1' and the initial guess at x_3, we solve the second equation for x_2:

$$x_2' = \frac{b_2 - a_{21}x_1' + a_{23}x_3}{a_{22}}$$

Finally, using the new approximation for x_1' and x_2', we solve the third equation for x_3:

$$x_3' = \frac{b_3 - a_{31}x_1' - a_{32}x'_2}{a_{33}}$$

This process of computing a new value for each of the variables constitutes one iteration. Now we perform another iteration, always using the most recently computed value of each variable. If the system of equations satisfies the criterion that the diagonal terms dominate their rows, the iteration scheme is guaranteed to converge to a solution, regardless of the initial guesses used.

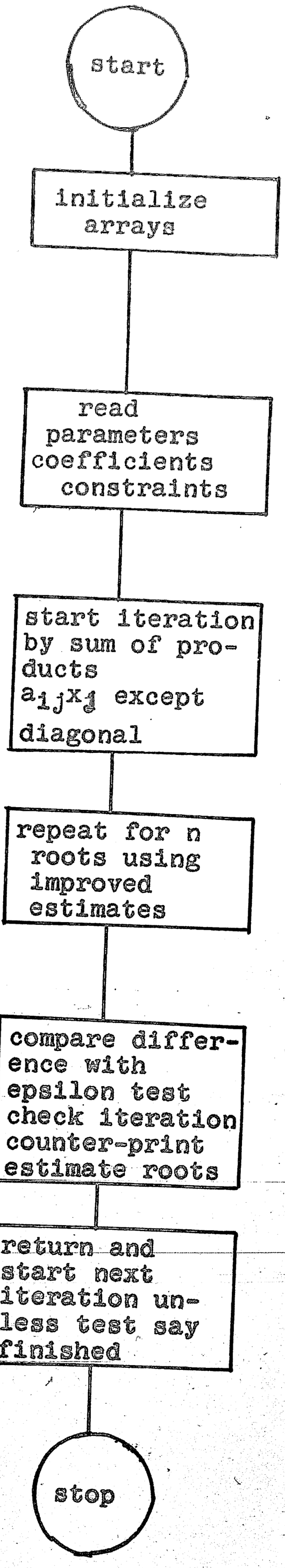
Statement of the Problem

We wish to write a program that will apply the Gauss-Siedel method to any system of up to 20 equations. The program must include the reading of the coefficients, of which there would be 2500 in the largest case, and the constant terms. Also included must be:

1. a test to determine if the condition of dominance is satisfied (stop and print a suitable message if it is not).
2. a test for convergence to decide when the approximations are close enough.
3. an iteration counter to stop the process if for any reason it fails to converge.

Output Format

```
* *GAUSS-SIEDEL METHOD FOR SOLVING SIMULTANEOUS EQUATIONS* *

NO. OF UNKNOWNS  xx

EPSILON  x.xxxxxxx

     I                    ROOT (I)
*******               *********

    xx              xx.xxxxxxxE-xx
     .                     .
     .                     .
     .                     .
     .                     .

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```



Basic Program Logic

N - number of unknowns
e - convergence test (epsilon)
M - maximum allowable no. of iterations

Columns	Content
1-2	Row number coefficient
3-4	Column number of coefficient
5-14	Coefficient
15	Sentinel: zero on all cards except last

Appendix 9

Batch Turnaround Time - Exponential (EX = 1 hour)

NO.	R	ALOG (R) f(R)	1-f(R)	TURNAROUND HRS:MIN.
1	93	9.93	.07	:04
2	85	9.85	.15	:09
3	22	8.47	1.53	1:32
4	90	9.89	.11	:07
5	85	9.84	.16	:10
6	61	9.51	.49	:29
7	81	9.79	.21	:13
8	07	7.34	2.66	2:40
9	19	8.34	1.66	1:40
10	92	9.92	.08	:05
11	61	9.51	.49	:29
12	91	9.91	.09	:05
13	64	9.55	.45	:27
14	27	8.69	1.31	1:19
15	44	9.18	.82	:49
16	76	9.73	.27	:16
17	78	9.75	.25	:15
18	78	9.75	.25	:15
19	27	8.69	1.31	1:19
20	74	9.70	.30	:18
21	30	8.80	1.20	1:12

Appendix 10

Listing of Diagnostic Error Messages for SDS 940 SKMA Computer - Time-Sharing System

ERRORS IN LIBRARY ROUTINES

1. ARGUMENT IN COMPLEX LIBRARY FUNCTION IS NOT REAL, INTEGER OR COMPLEX.
2. ARGUMENT IN EXTENDED PRECISION LIBRARY FUNCTION IS NOT REAL, INTEGER OR EXTENDED PRECISION.
3. ARGUMENT IN REAL LIBRARY FUNCTION IS NOT REAL, INTEGER OR EXTENDED PRECISION.
4. ERROR IN COMPLEX ARCTANGENT.
5. ERROR IN HYPERBOLIC FUNCTION.
6. EXPONENT TOO LARGE IN EXPF FUNCTION.
7. MORE THAN ONE ARGUMENT IN A LIBRARY FUNCTION REQUIRING ONLY ONE.
8. NEGATIVE ARGUMENT IN SQUARE ROOT.
9. NEGATIVE OR ZERO LOGARITHM.
10. NOT TWO ARGUMENTS IN A TWO ARGUMENT LIBRARY FUNCTION.
11. NUMBER TOO LARGE TO BE INTEGERIZED.

ERRORS IN I/O AND DATA STATEMENTS

1. A FORMATTED OPERATION TO A BINARY FILE OR A BINARY OPERATION TO A SYMBOLIC FILE.
2. ALPHABETIC DATA CAN ONLY BE STORED IN A REAL VARIABLE.
3. AN ATTEMPT HAS BEEN MADE TO OUTPUT A VARIABLE WHICH HAS NEVER BEEN STORED INTO.
4. BOOLEAN VARIABLES CAN BE OUTPUT ONLY WITH AN "A" FORMAT.
5. FIELD WIDTH GREATER THAN 63.
6. FIELD WIDTH LESS THAN NUMBER OF DIGITS AFTER DECIMAL POINT.
7. FILE CAN NOT BE OPENED - MAY ALREADY BE OPEN.
8. FILE DESIGNATOR IS NOT IN RANGE 0-4.
9. FILE HAS NEVER BEEN OPENED.
10. FILE NAME NOT IN DIRECTORY.
11. FORMAT CONTAINS A STRING HAVING AN ILLEGAL PRIME QUOTE COUNT.
12. FORMAT PART EXCEEDS I/O BUFFER SIZE.
13. FORMAT NOT A STRING VARIABLE.
14. ILLEGAL CHARACTER IN FORMAT.
15. ILLEGAL CHARACTER IN INPUT FIELD.

Listing of Diagnostic Error Messages for

SDS 940 SKMA Computer - Time-Sharing System

(continued)

16. ILLEGAL FORM OF SCALE FACTOR.
17. ILLEGAL HOLLERITH COUNT IN DATA STATEMENT.
18. ILLEGAL REPETITION NUMBER.
19. ILLEGAL RIGHT PARENTHESIS IN DATA STATEMENT.
20. ILLEGAL RIGHT PARENTHESIS IN FORMAT NEST.
21. INPUT FORM BINARY FILE TO A STRING OR TEXT VARIABLE NOT ALLOWED.
22. INPUT NUMBER IS TOO LARGE OR TOO SMALL.
23. INPUT STRING HAS AN ILLEGAL PRIME-QUOTE COUNT.
24. LOGICAL VARIABLE BEING INPUT WAS ALL BLANK.
25. LOGICAL VARIABLE DOES NOT START WITH T OR F.
26. NUMBER OF DIGITS AFTER DECIMAL POINT GREATER THAN
27. ONLY AN INTEGER CAN BE USED FOR A SCALE FACTOR.
28. ONLY THE CHARACTER "A" IS ALLOWED IN "A" FORMAT PART.
29. ONLY THE CHARACTER "O" IS ALLOWED IN "O" FORMAT PART.
30. OUTPUT OF A STRING OR TEXT VARIABLE TO A BINARY FILE NOT ALLOWED.
31. OVERFLOW OF DATA STATEMENT NESTING STACK.
32. OVERFLOW OF FORMAT NESTING STACK.
33. READ FROM A FILE OPENED FOR OUTPUT OR OUTPUT TO A FILE OPENED FOR INPUT.
34. SCALE FACTOR TOO LARGE.
35. STRING VARIABLES CAN BE OUTPUT ONLY WITH AN "A" FORMAT OR WITH FREE FORM OUTPUT.
36. TERMINAL DOLLAR SIGN IS MISSING FROM TEXT IN FORMAT.
37. UNEXPECTED END-OF-FILE.
38. ZERO COUNT FOR HOLLERITH FORMAT PART.

Listing of Diagnostic Error Messages for

SDS 940 SKMA Computer - Time-Sharing System
(continued)

MISCELLANEOUS ERRORS

1. A=0 AND B NON-POSITIVE IN A**B.
2. A IN A**B NOT TYPED REAL OR INTEGER.
3. A IS NEGATIVE AND B IS TYPED REAL IN A**B.
4. ATTEMPT TO DIVIDE BY ZERO.
5. ATTEMPTED ARITHMETIC WITH LOGICAL VALUE.
6. B IN A**B NOT TYPED REAL OR INTEGER.
7. CALL TO UNDEFINED EXTERNAL ROUTINE.
8. COMMON ILLEGALLY LENGTHENED BY "/'
9. DIMENSIONED VARIABLE HAS NO SUBSCRIPT.
10. DO INCREMENT APPEARS IN A DECLARATIVE STATEMENT.
11. DO MAXIMUM APPEARS IN A DECLARATIVE STATEMENT.
12. DO MINIMUM APPEARS IN A DECLARATIVE STATEMENT.
13. DO VARIABLE APPEARS IN A DECLARATIVE STATEMENT.
14. EXTERNAL ROUTINE NAME USED AS VARIABLE.
15. GO TO VARIABLE HAS NOT BEEN ASSIGNED TO A STATEMENT LABEL.
16. LABEL MISSING.
17. LOGICAL OPERATOR REQUIRES REAL OR INTEGER OPERANDS.
18. MULTIPLE DECLARATION OF "/'
19. NUMBER OF SUBSCRIPTS INCORRECT.
20. STORAGE CAPACITY EXCEEDED BY "/'
21. STORING LOGICAL VALUE INTO NON-LOGICAL VARIABLE.
22. STORING NON-LOGICAL VALUE INTO LOGICAL VARIABLE.
23. STRING AND TEXT FUNCTIONS NOT IMPLEMENTED.
24. SUBSCRIPT NOT TYPED REAL OR INTEGER.
25. SUBSCRIPT VALUE TOO LARGE.
26. SUBSCRIPT VALUE TOO SMALL.
27. SUBSCRIPTED VARIABLE NOT DIMENSIONED.
28. VALUE OF GO TO VARIABLE TOO LARGE.
29. VARIABLE HAS NOT BEEN ASSIGNED
30. $ LABEL IS MISSING.

Appendix 11

Diagnostic Error Measures -

Batch System IBM 1130

Error Number	Cause of Error
C00	WORKING STORAGE EXCEEDED The working storage area on disk is too small to accommodate the compiled program in disk system format.
C01	Non-numeric character in statement number.
C02	More than five continuation cards, or continuation card out of sequence.
C03	Syntax error in CALL LINK or CALL EXIT statement.
C04	Undeterminable, misspelled, or incorrectly formed statement.
C05	Statement out of sequence.
C06	Statement following STOP, RETURN, CALL LINK, CALL EXIT, GO TO, or IF statement does not have statement number.
C07	Name longer than five characters, or name not starting with an alphabetic character.
C08	Incorrect or missing subscript within dimension information (DIMENSION, COMMON, REAL, or INTEGER).
C09	Duplicate statement number.
C10	Syntax error in COMMON statement.
C11	Duplicate name in COMMON statement.
C12	Syntax error in FUNCTION or SUBROUTINE statement.
C13	Parameter (dummy argument) appears in COMMON statement.
C14	Name appears twice as a parameter in SUBROUTINE or FUNCTION statement.
C15	*IOCS control record in a subprogram.
C16	Syntax error in DIMENSION statement.
C17	Subprogram name in DIMENSION statement.
C18	Name dimensioned more than once, or not dimensioned on first appearance of name.
C19	Syntax error in REAL, INTEGER, or EXTERNAL statement.
C20	Subprogram name in REAL or INTEGER statement.
C21	Name in EXTERNAL that is also in a COMMON or DIMENSION statement.
C22	IFIX or FLOAT in EXTERNAL statement.
C23	Invalid real constant.
C24	Invalid integer constant.
C25	More than 15 dummy arguments, or duplicate dummy argument in statement function argument list.
C26	Right parenthesis missing from a subscript expression.
C27	Syntax error in FORMAT statement.
C28	FORMAT statement without statement number.
C29	Field width specification greater than 145.
C30	In a FORMAT statement specifying E or F conversion, w greater than 127, d greater than 31, or d greater than w, where w is an unsigned integer constant specifying the total field length of the data, and d is an unsigned integer constant specifying the number of decimal places to the right of the decimal point.
C31	Subscript error in EQUIVALENCE statement.
C32	Subscripted variable in a statement function.
C33	Incorrectly formed subscript expression.
C34	Undefined variable in subscript expression.
C35	Number of subscripts in a subscript expression does not agree with the dimension information.
C36	Invalid arithmetic statement or variable; or, in a FUNCTION subprogram the left side of an arithmetic statement is a dummy argument or in COMMON.
C37	Syntax error in IF statement.
C38	Invalid expression in IF statement.
C39	Syntax error or invalid simple argument in CALL statement.
C40	Invalid expression in CALL statement.
C41	Invalid expression to the left of an equal sign in a statement function.
C42	Invalid expression to the right of an equal sign in a statement function.
C43	In an IF, GO TO, or DO statement, a statement number is missing, invalid, incorrectly placed, or is the number of a FORMAT statement.
C44	Syntax error in READ or WRITE statement.
C45	*IOCS record missing with a READ or WRITE statement (mainline program only).
C46	FORMAT statement number missing or incorrect in a READ or WRITE statement.
C47	Syntax error in input/output list; or an invalid list element; or, in a FUNCTION subprogram, the input list element is a dummy argument or in COMMON.
C48	Syntax error in GO TO statement.
C49	Index of a computed GO TO is missing, invalid, or not preceded by a comma.
C50	*TRANSFER TRACE or *ARITHMETIC TRACE control record present, with no *IOCS control record in a mainline program.
C51	Incorrect nesting of DO statements; or the terminal statement of the associated DO statement is a GO TO, IF, RETURN, FORMAT, STOP, PAUSE, or DO statement.
C52	More than 25 nested DO statements.
C53	Syntax error in DO statement.
C54	Initial value in DO statement is zero.
C55	In a FUNCTION subprogram the index of DO is a dummy argument or in COMMON.
C56	Syntax error in BACKSPACE statement.
C57	Syntax error in REWIND statement.
C58	Syntax error in END FILE statement.
C59	Syntax error in STOP statement.
C60	Syntax error in PAUSE statement.
C61	Integer constant in STOP or PAUSE statement is greater than 9999.
C62	Last executable statement before END statement is not a STOP, GO TO, IF, CALL LINK, CALL EXIT, or RETURN statement.
C63	Statement contains more than 15 different subscript expressions.
C64	Statement too long to be scanned, because of compiler expansion of subscript expressions or compiler addition of generated temporary storage locations.
C65*	All variables are undefined in an EQUIVALENCE list.

Diagnostic Error Measures -

Batch System IBM 1130

(continued)

Error Number	Cause of Error
C66*	Variable made equivalent to an element of an array in such a manner as to cause the array to extend beyond the origin of the COMMON area.
C67*	Two variables or array elements in COMMON are equated, or the relative locations of two variables or array elements are assigned more than once (directly or indirectly).
C68	Syntax error in an EQUIVALENCE statement; or an illegal variable name in an EQUIVALENCE list.
C69	Subprogram does not contain a RETURN statement, or a mainline program contains a RETURN statement.
C70	No DEFINE FILE statement in a mainline program that has disk READ, WRITE, or FIND statements.
C71	Syntax error in DEFINE FILE statement.
C72	Duplicate DEFINE FILE statement, more than 75 DEFINE FILES, or DEFINE FILE statement in subprogram.
C73	Syntax error in record number of disk READ, WRITE, or FIND statement.
C75	Syntax error in DATA statement.
C76	Names and constants in a DATA statement not in a one to one correspondence.
C77	Mixed mode in DATA statement.
C78	Invalid hollerith constant in a DATA statement.
C79	Invalid hexadecimal specification in a DATA statement.
C80	Variable in a DATA statement not used elsewhere in the program.
C81	COMMON variable loaded with a DATA specification.
C82	DATA statement too long.

Accumulator Display	Cause of Error
F000	No *IOCS card appeared with the mainline program and I/O was attempted in a subroutine.
F001	Logical unit defined incorrectly, or No *IOCS control record for specified I/O device.
F002	Requested record exceeds allocated buffer size.
F003	Illegal character encountered in input record.
F004	Exponent too large or too small in input field.
F005	More than one E encountered in input field.
F006	More than one sign encountered in input field.
F007	More than one decimal point encountered in input field.
F008	Read of output-only device, or Write of input-only device.
F009	Real variable transmitted with an I format specification or integer variable transmitted with an E or F format specification.
F020	Illegal unit reference.*
F021	Read list exceeds length of write list.*
F022	Record does not exist for read list element.*
F023	Maximum length of $$$$$ area on the disk has been exceeded. This error is unrecoverable and results in a CALL EXIT.*
F100	File not defined by DEFINE FILE statement.
F101	File record too large, equal to zero, or negative.
F103	Disk FIO (SDFIO) has not been initialized.
F10A	Subscripting has destroyed the Define File Table. This occurs when a subscript exceeds the specification in a DIMENSION statement.

*) Can occur in unformatted I/O operations.

Appendix 12

Example of Bartlett's Test for Homoscedasticity

To illustrate Bartlett's test consider the following example:

For the analysis of the occurrence and persistence of program errors as measured by the maximum number of error run units associated with the highest error of that group, we have:

Maximum Error Run Units: System (system column total - all cause)

We have from the experiment:

Entry	No. of Data Value	Mean	Deviation (5)
Batch Problem 1	4	6.0	1.5
Batch Problem 2	4	4.75	.5448
Conversational Problem 1	4	6.5	1.75
Conversational Problem 2	4	6.5	2.2776

To use Bartlett's test we first derive $\sum_i n_i \log s_i^2$

where $s^2 = N\sigma^2/(N-1) = \sum (x-\bar{x})^2/(N-1)$

$$3 \log (1.5)^2 = 3\ (.352)$$

$$3 \log (.5448)^2 = 3\ (9.473 - 10)$$

$$3 \log (1.75)^2 = 3\ (.485)$$

$$3 \log (2.2776)^2 = \frac{3\ (716)}{3\ (1.026)} = 3.078$$

then we compute $n \log\left(\sum n_i s_i^2/n\right)$

$$= 12 \log \left\{\sum s_i^2 / (n_i/n)\right\}$$

$$= 12 \log \left\{\sum s_i^2/4 = 12 \log 2.7 = 12\,(.431)\right\}$$

$$= 5.175$$

Now we have

$$M = 2.3026 \quad (5.175 - 3.078) = 4.83$$

$$C = 1 + \frac{1}{9}\left(\frac{4}{3} - \frac{1}{12}\right) = 1.139$$

$$M/C = \frac{4.83}{1.139} = 4.235$$

The chi-square value for the .05 level ($\chi^2 0.05$) for $n = 4-1 = 3$ degrees of freedom is 7.82. Since our M/C is less than this, the null hypothesis that the variance all come from the same universe is <u>not</u> rejected. We conclude that the variability of the process is constant.

Example of Aspin-Welch Test

To illustrate the use of the Aspin-Welch test for differences between two sample means consider again the results of the Maximum Error Run Units figures for the detection level - system (over all causes) as shown below:

Maximum Error Run Units (Systems)

Subjects	Problem	Mode	M.E.R.U.	
1	1	B	5	Set x_1
2	1	B	11	
3	1	B	3	
4	1	B	5	
5	2	B	5	
6	2	B	6	
7	2	B	3	
8	2	B	5	
1	2	C	3	Set x_2
2	2	C	10	
3	2	C	3	
4	2	C	10	
5	1	C	6	
6	1	C	4	
7	1	C	2	
8	1	C	14	

Supposing we are unwilling to assume that the same variation exists under both experimental modes. Then we can test the hypothesis that $\bar{X}_1' = \bar{X}_2'$ as follows:

$$\bar{X}_1 = \frac{5.0\ 11.0\ 3.0\ 5.0\ 5.0\ 6.0\ 3.0\ 5.0}{8} = 5.37$$

$$\bar{X}_2 = \frac{3.0\ 10.0\ 3.0\ 10.0\ 6.0\ 4.0\ 2.0\ 14.0}{8} = 6.50$$

Next we have $s_1^2 = \dfrac{\sum(x_1-\bar{x}_1)^2}{N_1-1} = \dfrac{\sum x_1^2 - \left(\sum x_1\right)^2/N_1}{N_1-1}$

$$s_1^2 = \frac{(5)^2\ (11)^2\ (3)^2\ (5)^2\ (6)^2\ (3)^2\ (5)^2 - (43)2/8}{7}$$

$$= 4.86$$

$$s_2^2 = \sum(x_2-\bar{x}_2)^2$$

$$= \frac{\sum x_2^2 - \left(\sum x_2\right)^2/N_2}{N_2-1}$$

$$s_2^2 = \frac{(3)^2\ (10)^2\ (3)^2\ (10)^2\ (6)^2\ (4)^2\ (2)^2\ (14)^2 - (52)2/8}{7}$$

$$= 18.85$$

now $$c = \frac{s_1^2/N_1}{s_1^2/N_1 + s_2^2/N_2}$$

$$= \frac{4.86/8}{\frac{4.86/8}{\ } \quad \frac{18.85/8}{\ }}$$

$$= .205$$

Hence, the degrees of freedom are given by:

$$n = \left\{ \frac{1}{\frac{c^2}{N_1 - 1} + \frac{(1 - c)^2}{N_2 - 1}} \right\}$$

$$= \frac{1}{\frac{(.205)^2}{7} + \frac{(.795)^2}{7}}$$

$$= 10.3 \cong 10$$

Hence the test is: $$t = \frac{\bar{x}_1 - \bar{x}_2}{\left(s_1^2/N_1 + s_2^2/N_2\right)^{1/2}}$$

$$t = \frac{6.50 - 5.37}{\left(4.86/8 + 18.85/8\right)^{1/2}}$$

$$t = .657$$

which is less than the $t_{.05}$, 10 points = 2.228. In fact $t_{.10,10} = 1.812$ and even at 20%, $t_{.02,10} = 1.372$. We, therefore, accept the typothesis that $\bar{x}_1'$ and $\bar{x}_2'$ equal, and thus we cannot discriminate between the two programming modes, batch and conversational time-sharing. Note particularly that we lost four degrees of freedom as compared to the ordinary t test.

Bibliography

1. Adams, Jeanne and Cohen, Leonard, "Time-Sharing vs. Instant Batch Processing; An Experiment in Programmer Training", Computers and Automation, Vol. 18, No. 3, March 1969.

2. Colilla, Robert A., "Time-Sharing and Multiprocessing Terminology", Datamation, March 1966.

3. Duncan, Acheson J., Quality Control and Industrial Statistics, 3rd Edition, Richard D. Irwin, Inc., Homewood, Illinois, 1967.

4. Gold, M., Methodology for Evaluating Time-Shared Computer Usage, Doctoral dissertation, MIT, Sloan School for Management, 1967.

5. Grant, E. E., Sackman H., An Exploratory Investigation of Programmer Performance Under On-Line and Off-Line Conditions, SP 2581 SDC, Santa Monica, Calif., Sept. 1966.

6. Hyman, Harris, "The Time-Sharing Business", Datamation, February 1967.

7. Johnson, N. C., Leone, F. C., Statistical and Experimental Design: Volume II, Wiley & Sons, New York 1964.

8. Nielsen, Norman R., "The Simulation of Time-Sharing", Datamation, July, 1967.

9. O'Sullivan, T. C., "Terminal Networks for Time-Sharing", Datamation, July, 1967.

10. Patrick, Robert L., "Time-Sharing Tally Sheet", Datamation, November, 1967.

11. Ryan, J. L., Crandall, R. L., Medwedeff, M. C., "A Conversational System for Incremental Compilation and Execution in a Time-Sharing Environment", Proceedings--Fall Joint Computer Conference 1966.

Bibliography

12. Sackman H., Erikson, W. J., Grant, E. E., "Exploratory Experimental Studies Comparing On-Line and Off-Line Programming Performances", Communications of the ACM, January, 1968, Vol. II, No. 1 (SDC).

13. Sackman, Harold, Computer, System Science and Evolving Society, John Wiley & Sons, Inc., New York, 1967.

14. Sackman, Harold, Time-Sharing Versus Batch Processing, the Experimental Evidence, SDC, Spring Joint Computer Conference 1968.

15. Sackman, H., Gold, Jr., Time-Sharing Versus Batch Processing: An Experimental Inquiry into Human Problem Solving, SDC, SP-3110, June 1968.

16. Schatzoff, M., Tsao R., Wiig R., "An Experimental Comparison of Time-Sharing and Batch Processing", Communication of the ACM, May 1967, Vol. 10, No. 5 (IBM).

17. Scherr, Allan L., "Time-Sharing Measurement", Datamation, April, 1966.

18. Smith, Lyle B., "A Comparison of Batch Processing and Instant Turnaround", Communications of the ACM, August, 1967, Vol. 10, No. 8 (Stanford).

19. Smith, Wm. A. Jr., Nature and Detection of Errors in Production Data Collection, Doctoral dissertation, NYU 1966, New York.

20. Waks, David J., "Conversational Computing on a Small Machine", Datamation, April, 1967.

Vita

Donald M. Jackson was born in Waukegan, Illinois, in 1942, the son of Hugh and Ethel Jackson. He received his secondary education in Freeport, New York, graduating in the upper quarter of his class in 1960. He then attended Lehigh University for four years and was conferred a Bachelor of Science degree in Industrial Engineering. After three years of industrial employment with Gulf Oil Company, he returned to Lehigh University for graduate studies and an Instructors teaching position. His publications include articles in the National Petroleum Refiners Association and the American Institute of Industrial Engineers Journal. Most recently in the AIIE Journal, he co-authored with D. R. Zerbe, "Determination of Standard Sizes for Manufacturing Using Dynamic Programming", AIIE Journal, August 1968, Vol. 19, No. 6.