National Bureau
of Standards

Computer Science and Technology

NBS Special Publication 500-104

Proceedings of the Computer Performance Evaluation Users Group 19th Meeting

CPEUG83

"CPE - A NEW PERSPECTIVE:
The impact of the technology
revolution."

NATIONAL BUREAU OF STANDARDS

The National Bureau of Standards[1] was established by an act of Congress on March 3, 1901. The Bureau's overall goal is to strengthen and advance the Nation's science and technology and facilitate their effective application for public benefit. To this end, the Bureau conducts research and provides: (1) a basis for the Nation's physical measurement system, (2) scientific and technological services for industry and government, (3) a technical basis for equity in trade, and (4) technical services to promote public safety. The Bureau's technical work is performed by the National Measurement Laboratory, the National Engineering Laboratory, and the Institute for Computer Sciences and Technology.

THE NATIONAL MEASUREMENT LABORATORY provides the national system of physical and chemical and materials measurement; coordinates the system with measurement systems of other nations and furnishes essential services leading to accurate and uniform physical and chemical measurement throughout the Nation's scientific community, industry, and commerce; conducts materials research leading to improved methods of measurement, standards, and data on the properties of materials needed by industry, commerce, educational institutions, and Government; provides advisory and research services to other Government agencies; develops, produces, and distributes Standard Reference Materials; and provides calibration services. The Laboratory consists of the following centers:

Absolute Physical Quantities[2] — Radiation Research — Chemical Physics — Analytical Chemistry — Materials Science

THE NATIONAL ENGINEERING LABORATORY provides technology and technical services to the public and private sectors to address national needs and to solve national problems; conducts research in engineering and applied science in support of these efforts; builds and maintains competence in the necessary disciplines required to carry out this research and technical service; develops engineering data and measurement capabilities; provides engineering measurement traceability services; develops test methods and proposes engineering standards and code changes; develops and proposes new engineering practices; and develops and improves mechanisms to transfer results of its research to the ultimate user. The Laboratory consists of the following centers:

Applied Mathematics — Electronics and Electrical Engineering[2] — Manufacturing Engineering — Building Technology — Fire Research — Chemical Engineering[2]

THE INSTITUTE FOR COMPUTER SCIENCES AND TECHNOLOGY conducts research and provides scientific and technical services to aid Federal agencies in the selection, acquisition, application, and use of computer technology to improve effectiveness and economy in Government operations in accordance with Public Law 89-306 (40 U.S.C. 759), relevant Executive Orders, and other directives; carries out this mission by managing the Federal Information Processing Standards Program, developing Federal ADP standards guidelines, and managing Federal participation in ADP voluntary standardization activities; provides scientific and technological advisory services and assistance to Federal agencies; and provides the technical foundation for computer-related policies of the Federal Government. The Institute consists of the following centers:

Programming Science and Technology — Computer Systems Engineering.

[1] Headquarters and Laboratories at Gaithersburg, MD, unless otherwise noted;
mailing address Washington, DC 20234.

[2] Some divisions within the center are located at Boulder, CO 80303.

Computer Science
and Technology

NBS Special Publication 500-104

Proceedings of the Computer Performance Evaluation Users Group (CPEUG) 19th Meeting

San Francisco, California
October 25 - 28, 1983

Proceedings Editor
Deborah Mobray

Conference Host
Navy Regional Data Automation Center
Department of the Navy

Sponsored by
Institute for Computer Sciences and Technology
National Bureau of Standards
Washington, DC 20234

U.S. DEPARTMENT OF COMMERCE
Malcolm Baldrige, Secretary

National Bureau of Standards
Ernest Ambler, Director

Issued October 1983

Reports on Computer Science and Technology

The National Bureau of Standards has a special responsibility within the Federal Government for computer science and technology activities. The programs of the NBS Institute for Computer Sciences and Technology are designed to provide ADP standards, guidelines, and technical advisory services to improve the effectiveness of computer utilization in the Federal sector, and to perform appropriate research and development efforts as foundation for such activities and programs. This publication series will report these NBS efforts to the Federal computer community as well as to interested specialists in the academic and private sectors. Those wishing to receive notices of publications in this series should complete and return the form at the end of this publication.

Library of Congress Catalog Card Number: 82-600594

National Bureau of Standards Special Publication 500-104
Natl. Bur. Stand. (U.S.), Spec. Publ. 500-104, 236 pages (Oct. 1983)
CODEN: XNBSAV

U.S. GOVERNMENT PRINTING OFFICE
WASHINGTON: 1983

For sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402
Price $6.50
(Add 25 percent for other than U.S. mailing)

FOREWORD

The data processing environment has undergone profound changes since CPEUG was founded in 1971. Microcomputers, at that time existed only in specialized process control applications. Now end-user microcomputers are commonplace and their users have expectations unheard of twelve years ago. In parallel has been the increasing perception of traditional large, mainframe data processing as only one aspect of a broader view of information resources and information technology. With these changes, the challenge of providing efficient and effective user support has also grown.

It is with these changes in mind that this year's CPEUG conference has chosen as a theme "CPE-A New Perspective: The impact of the technology revolution." This year's conference offers topics ranging from microcomputers to supercomputers. The increasingly complex area of data communications is presented as well as topics in office automation, software improvement and engineering, capacity planning, and quality assurance, to mention just a few. The diversity of topics reflects the broad range of areas which CPE analysts must now consider.

The challenges inherent in the increasingly complex areas of information technology also provide new opportunities to increase the effectiveness of the services we provide. Even as the technology has grown so has the volume and type of information which the users wish to store, manipulate, and retrieve. We must be knowledgeable in many new areas to ensure that we are using not only the most efficient means available but also the must effective.

This year's CPEUG topics were specifically chosen to reflect the breadth of the CPE field. I believe you will enjoy them as well as learn from them.

JOHN CARON
CPEUG 83 Conference Chairperson

PREFACE

The theme of CPEUG 83, "CPE - A NEW PERSPECTIVE: The impact of the technology revolution," focuses on the rapid introduction of sophisticated end-user technology, and addresses the impact of this revolution on CPE and the CPE professional. The debate over the question raised several years ago by a former CPEUG Program Chairman: "How will CPE, traditionally associated with large central computers, change in an era of smaller, cheaper hardware and improved digital communications?" intensifies. The CPE professional is challenged to react to the new technology and trends. The keynote address, "Microcomputers: The Risks and Rewards" and the keynote panel "Information System Cost Performance -- New Directions" highlights and sets the framework for this challenge and debate. The conference focuses on the integration of user microcomputer systems into the overall ADP structure and concentrates on micros and end user computing activities -- "Strengthening the End-User Interface", "Managing End-User Computing", for example. The growing relative importance of software is recognized and software related issues are addressed in several sessions.

As in the previous Conferences, tutorials, case studies, panels, and technical sessions are included in the program. There are several sessions that address the information needs of the first-time attendees, experienced CPE analysts, managers, and interested data processing professionals. The CPEUG Conference is one of the indispensable events on the calendar of computer performance professionals. CPEUG 83 provides a forum for shaping subsequent Conferences throughout the 1980's.

The CPEUG 83 program was the work of many people. Paul Roth, Vice Chairperson for academia; Jim Sprung Vice Chairperson for industry; Arnold Johnson, Vice Chairperson for Government; and Dr Deborah Mobray, Proceedings Editor, were this year's vital links to broaden program participation. The Conference Committee, session chairpersons, authors, and tutors all deserve recognition for their time, patience, and participation. The invaluable support of Sylvia Mabie merits special thanks.

CHARLES A. SELF
CPEUG 83 Chairperson
October 1983

ABSTRACT

These Proceedings record the papers that were presented at the Nineteenth Meeting of the Computer Performance Evaluation Users Group (CPEUG 83) held October 25-28, 1983 in San Francisco, CA. CPEUG 83 recognized the rapid introduction of sophisticated end-user technology into the information processing environment and addressed the challenges posed to the CPE community. CPEUG 83 offered topics ranging from microcomputers to supercomputers. The increasingly complex area of data communications was presented as well as topics in office automation, software improvement and engineering, capacity planning, and quality assurance. The program was divided into three parallel sessions and included technical papers, case studies, tutorials, and panels. Technical papers are presented in the Proceedings in their entirety.

Key words: acquisition, benchmarking, capacity planning, computer performance evaluation, configuration management/quality assurance, cost accounting and chargeback, data communications, end-user computing, local area networks, microcomputers, modeling techniques, office automation, performance management, software engineering, and software improvement.

The material contained herein is the viewpoint of the authors of specific papers. Publication of their papers in this volume does not necessarily constitute an endorsement by the Computer Performance Evaluation Users Group (CPEUG) or the National Bureau of Standards. The material has been published in an effort to disseminate information and to promote the state-of-the-art of computer performance measurement, simulation, and evaluation.

CPEUG Advisory Board

CONFERENCE COMMITTEE

CONFERENCE CHAIRPERSON	John Caron General Services Administration (703) 756-6151
PROGRAM CHAIRPERSON	Charles Self Federal Computer Performance Evaluation & Simulation Center (FEDSIM) (703) 274-7045
PROGRAM VICE-CHAIRPERSON FOR FEDERAL GOVERNMENT	Arnold Johnson Federal Software Testing Center (703) 756-6153
PROGRAM VICE-CHAIRPERSON FOR INDUSTRY	Jim Sprung MITRE Corporation (703) 827-6446
PROGRAM VICE-CHAIRPERSON FOR ACADEMIA	Paul Roth Virginia Polytechnical Institute (703) 698-6023
PUBLICATIONS CHAIRPERSONS	Ken Moore Federal Reserve Board (202) 452-2832 Helen Letmanyi National Bureau of Standards (301) 921-3485
PUBLICITY CHAIRPERSON	Barbara Anderson Information Systems and Networks Corporation (703) 694-8180
AWARDS CHAIRPERSON	Dennis Shaw US General Accounting Office (202) 275-6187
REGISTRATION CHAIRPERSON	Hal Hawes Federal Computer Performance Evaluation and Simulation Center (FEDSIM) (703) 274-7910
PROCEEDINGS EDITOR	Deborah Mobray Veterans Administration (202) 389-2646

NATIONAL ARRANGEMENTS CHAIRPERSON	Art Chantker Department of Transportation (202) 426-0945
FINANCE CHAIRPERSON	Tom Buschbach Federal Conversion Support Center (703) 756-6156
VENDOR EXHIBITS CHAIRPERSON	Helen McEwan Federal Software Exchange (703) 756-6150
LOCAL HOST	Robert Taylor Navy Regional Automation Center (415) 869-5202

TABLE OF CONTENTS

LOCAL AREA NETWORKS

MODELLING TECHNIQUES

SOFTWARE IMPROVEMENT

CAPACITY PLANNING

SOFTWARE ENGINEERING

MICROS AND END-USER COMPUTING

COST ACCOUNTING AND CHARGEBACK

CONFIGURATION MANAGEMENT

PERFORMANCE MANAGEMENT

BENCHMARKING

PROGRAM

MONDAY, OCTOBER 24

7:00 PM to 10:00 PM SOCIAL HOUR AND REGISTRATION

TUESDAY, OCTOBER 25

8:00 AM COFFEE AND REGISTRATION

10:00 AM WELCOMING ADDRESSES:
John Caron
Office of Software Development and Information Technology
General Services Administration
Washington, DC

Robert Taylor
Navy Regional Data Automation Center
San Francisco, CA

10:15 AM PROGRAM OVERVIEW
Charles A. Self
Federal Computer Performance Evaluation and Simulation Center (FEDSIM)
Washington, DC

10:30 AM KEYNOTE ADDRESS: Microcomputers -- The Risks and Rewards
Thomas Willmott
International Data Corp
Framingham, MA

11:30 AM CONFERENCE LUNCHEON AND SPEAKER: Supercomputer Architectures
John Machado
U.S. Navy
Washington, DC

2:30 PM KEYNOTE PANEL: Information System Cost Performance -- New Directions
Chairperson: Thomas F. Wyrick, Manager
Telecommunicatons Modeling and Analysis
GTE Service Corporation
Tampa, FL

Carl Palmer, Deputy for Operations
Information Management and Technology
Division
General Accounting Office
Washington, DC

Thomas Bell, President
Computer Technology Group
Palos Verdes Estates, CA

David R. Vincent
General Manager
Institute for Software Engineering
Sunnyvale, CA

5:00 PM VENDOR EXHIBITS AND COCKTAIL RECEPTION

TRACK A PROGRAM

WEDNESDAY, OCTOBER 26

8:00 AM COFFEE AND REGISTRATION

9:00 AM SESSION: Local Area Networks
Chairperson: Ron Rutledge
Department of Transportation
Cambridge, MA

Flow Control Performance and LAN's
Bart Stuck, Bell Laboratories
Murray Hill, NJ

Modeling and Monitoring a LAN
W. Bruce Watson
Lawrence Livermore National Laboratory
Livermore, CA

Performance Evaluation of LAN's in the Government
Ken J. Thurber and Harvey Freeman
Architecture Technology Corporation
Minneapolis, MN

Performance Considerations for Personal Computers and LAN's
Ken Omahen
Digital Equipment Corporation
Maynard,MA

12:30 PM CONFERENCE LUNCHEON AND SPEAKER: CPE and CPEUG -- Another Look
Paul Roth
Virginia Polytechnical Institute
Blacksburg, VA

3:00 PM SESSION: Modeling Techniques
Chairperson: Charles Shub
American Bell
Denver, CO

Queue Length Characteristics at Very Fast, Constant Service Time Merger Nodes
Chaim Ziegler
Brooklyn College
Brooklyn, NY

AISIM: An Interactive Graphics Discret Event Simulation Tool
Herman Schultz
MITRE Corporation
Bradford, MA

William Letendre
US Air Force Electronics Systems Division
Hanscom AFB, MA

The Application of Multivariate
Statistical Techniques to Computer
Performance Evaluation Using Simulated
Data
Thomas Hartrum
Air Force Institute of Technology
Wrigth-Patterson AFB, OH

Improving the Accuracy of a
Working-Set-Oriented
Generative Model of Program Behavior
Domenico Ferrari
Tzong-Yu Paul Lee
University of California
Berkley, CA

5:00 PM VENDOR EXHIBITS AND COCKTAIL RECEPTION

THURSDAY, OCTOBER 27

8:00 AM COFFEE AND REGISTRATION

9:00 AM SESSION: Managing End-User Computing
Chairperson: Tom Pyke
National Bureau of Standards
Gaithersburg, MD

The Evaluation of the Information Resource
Center Concept
Tom Pyke
National Bureau of Standards
Gaithersburg, MD

Information Center: The Users Answer to the Computer
Room
Esther Georgatos
Veterans Administration
Washington, DC

Support Structures: A Management Tool
Lawrence K. Berenson
Department of Interior
Washington, DC

11:00 AM SESSION: Micros and the New CPE Environment
Chairperson: Dennis M. Gilbert
National Bureau of Standards
Gaithersburg, MD

An Organizational Model and Case Study for
Microcomputer CPE
Malcom Campbell
State of Missouri
Jefferson City, Missouri

System Trends (Integrated
Environments/Interfacing Micros)
(To be Announced)

Micros and Converging Technologies
(To be Announced)

12:30 PM CONFERENCE LUNCHEON AND AWARDS PRESENTATION

3:00 PM SESSION: Federal Microcomputer Activities
Chairperson: Al Hankinson
National Bureau of Standards
Gaithersburg, MD

National Bureau of Standards Microcomputer
Products and Activities
Dennis M. Gilbert
National Bureau of Standards
Gaithersburg, MD

GSA Small System Initiatives
Ralph Simmons
General Services Administration
Washington, DC

A Government-wide Approach to Individual and
Organizational Productivity
Al Duncan
Department of Transportation, Inspector
General
Washington, DC

FRIDAY, OCTOBER 28

9:00 AM SESSION: Office Automation
Chairperson: Larry Jackson
MITRE Corporation
Mclean, VA

Assessing the Impact of Organizational and Human
Factors During Office Automation Implementations
Bob Kovach and Paul Chandler
MITRE Corporation
McLean, VA

GSA End-User Computer Pilot Project
Jerry Whiting
General Services Administration
Washington, DC

11:00 AM SESSION: Strengthening the End-User Interface

Chairperson: Thomas H. Acklen
Veterans Administration
Austin, TX

Information Centers: Creating and Supporting Independent Users
Trish Fineran
United States Department of Agriculture
Information Technology Center
Washington, DC

Data Processing Users Service - A Problem; A Proposed Solution
Thomas H. Acklen
Veterans Administration
Austin, TX

Microcomputer Policy: Enhancing End-User Productivity
Donald E. Humphries
National Oceanic Atmospheric Administration
Washington, DC

TRACK B PROGRAM

WEDNESDAY, OCTOBER 26

9:00 AM PANEL: CPE in the University?
Chairperson: Paul Roth
Virginia Polytechnical Institute
Blacksburg, VA

PANEL MEMBERS: Paul Roth
Virginia Polytechnical Institute
Blacksburg, VA

Charles Shub
American Bell
Denver, CO

Domenico Ferrari
University of California
Berkeley, CA

Bernie Domanski
City College of New York
New York, NY

11:00 AM SESSION: Benchmarking
Chairperson: Robert E. Waters
Air Force Computer Acquisition Center
Hanscom AFB, MA

Benchmark and Conversion Tool:
Test Data Reduction
Frances A. Kazlauski
Naval Data Automation Command
Washington, DC

System Integration and Test Using an
Automated Test Support System
TRW
Redondo Beach, CA

Benchmark Strategies
Robert E. Waters
Air Force Computer Acquisition Center
Hanscom AFB, MA

The Live Test Demonstration Manual - A
Format Tutorial
Lt Anthony F. Burnetto
Air Force Computer Acquisition Center
Hanscom AFB, MA

3:00 PM SESSION: Software Improvement
Chairperson: Carol Houtz
Federal Conversion Support Center
Washington, DC

Software Improvement Program
Opal A. Stroup
Defense Mapping Agency
US Naval Observatory
Washington, DC

Software Alternatives Available to Federal Agencies
Steven Merritt
General Accounting Office
Washington, DC

Impact of Software Obsolescence in Federal ADP Operations
Steven Merritt
General Accounting Office
Washington, DC

Software Improvement Program (SIP): A Treatment for Software Senility
Carol A. Houtz
Federal Conversion Support Center
Washington, DC

Software Improvement Through Automated Normalization
Michael G. Walker
WBG, Inc
McLean, VA

THURSDAY, OCTOBER 27

9:00 AM SESSION: Capacity Planning
Chairperson: James Sprung
MITRE Corporation
McLean, VA

Algebraic Models for CPU Sizing
Robert A. Orchard
Bell Laboratories
Whippany, NJ

An Approach to Capacity Management
James Weatherbee
Federal Computer Performance Evaluation and Simulation Center (FEDSIM)
Washington, DC

Specifying Computer Hardware Requirements for Planning
James Sprung, Ray Komajda, and Connie
Kulik
MITRE
McLean, VA

11:00 AM SESSION: Configuration Management/Quality Assurance
Chairperson: Barbara Christoph
MITRE Corporation
McLean, VA

Automating Configuration Management
Craig Riesberg and Lt DeJesus
HQMAC/ADCI
United States Air Force
Scott AFB, IL

Quality Assurance Overview
Margaret Brouse
MITRE Corporation
McLean, VA

3:00 PM SESSION: Software Engineering
Chairperson: George N. Baird
Federal Conversion Support Center
Washington, DC

Establishing a Software Engineering Technology (SET)
L. Arnold Johnson and William Milligan
Federal Conversion Support Center
Washington, DC

Software Testing Techniques Used by Federal Agencies
James V. Rinaldi, Jr.
General Accounting Office
Washington, DC

Characteristics of Software Development Team Structures
and Their Impact on Software Development
Ann von Mayrhauser
Illinois Institute of Technology
Chicago, IL

An Engineering Approach to Software Testing
Gary Fisher
Federal Conversion Support Center
Washington, DC

FRIDAY, OCTOBER 28

9:00 AM SESSION: Performance Management
Chairperson: Donald R. Deese
Federal Computer Performance Evaluation and Simulation Center (FEDSIM)
Washington, DC

The Terminal Probe Method Revisited. Some Statistical Considerations
Luis Felio Cabrera
Catholic University of Chile
Santiago, Chile

Some Elements of Software Functions and Cost Analysis as Related to Performance
John E. Gaffney, Jr
International Business Machines
Gaithersburg, MD

Modeling and Measuring to Improve Network Cost Performance
Ronald K. Leighton
GTE Services Corporation
Tampa, FL

11:00 AM SESSION: Cost Accounting and Chargeback
Chairperson: Dean Halstead
Federal Computer Performance Evaluation and Simulation Center (FEDSIM)
Washington, DC

Standard Costing for ADP Services
David R. Vincent
Institute for Software Engineering
Sunnyvale, CA

Cost and Revenue System at Parklawn Data Center
Tim Carrico
Parklawn Data Center
Health and Human Services
Rockville, MD

Developing a DP Charging System for the FAA
Harvey Kaplan
Federal Aviation Administration
Washington, DC

TRACK C PROGRAM

WEDNESDAY, OCTOBER 26

9:00 AM TUTORIAL: Requirements Analysis and Workload Characterization

Carl Palmer
General Accounting Office
Washington, DC

10:00 AM TUTORIAL: Use of Benchmarking in the Federal ADP Procurement Process

Dennis Shaw
General Accounting Office
Washington, DC

11:00 AM TUTORIAL: Software Development Guidelines

Anneliesse von Mayrhauser
Illinois Institute of Technology
Chicago, IL

3:00 PM TUTORIAL: Microcomputer Software

John Caron
Office of Software Development and
Information Technology
General Services Administration
Washington, DC

4:00 PM TUTORIAL: Usability

David F. Stevens
Lawrence Berkeley Laboratory
University of California
Berkeley, CA

THURSDAY, OCTOBER 27

9:00 AM TUTORIAL: IRM Planning

Nancy Doane
Federal IRM Planning Support Program
Washington, DC

10:00 AM TUTORIAL: Computer Service Selection

Anneliese von Mayrhauser
Dennis Erwin Witte
Illinois Institute of Technology
Chicago, IL

11:00 AM TUTORIAL: Security Certification of Applications Software

Fred Tompkins
MITRE
McLean, VA

3:00 PM TUTORIAL: ACMS: A Computer Modeling System

Duane Ball
Federal Computer Performance Evaluation
and Simulation Center (FEDSIM)
Washington, DC

4:00 PM TUTORIAL: Software Instrumentation Points

James P. Bouhana
Wang Institute of Graduate Studies
Tyngsboro, MA

FRIDAY, OCTOBER 28

9:00 AM TUTORIAL: Implementing a DP Chargeback System

Dean Halstead
Federal Computer Performance Evaluation
and Simulation Center (FEDSIM)
Washington, DC

Theoretical Performance Analysis of Virtual Circuit LAN Sliding Window Flow Control

E.Arthurs
G.L.Chesson
B.W.Stuck

Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

A transmitter breaks a message up into packets and transmits the packets to a receiver over a single virtual circuit within a local area network. The receiver has a finite amount of storage capacity for buffering messages. A *sliding window protocol* turns the transmitter on and off to insure there is always storage room in the receiver for packets. Mean throughput rate and delay statistics are studied as a function of model parameters.

1. Introduction

Our purpose is to study the traffic handling characteristics of a policy for pacing the flow of information from a transmitter to a receiver over a logical abstraction of a physical circuit (a so called *virtual* circuit) within a local area network. If the receiver has only a limited amount of storage, and the transmitter is faster than the receiver, messages can be transmitted and blocked or rejected by the receiver because no room is available. A protocol is used for controlling the flow of information to insure that no packet is lost due to no room being available, as well as for other reasons. Since the transmitter must be turned on and off to insure no messages are lost, what is the penalty in mean throughput rate and message delay statistics as a function of receiver buffering and transmitter and receiver processing speeds? References on this subject are found elsewhere (e.g., Tanenbaum, 1981, pp.187-196).

Examples of such mechanisms are stop-start protocols where the transmitter stops until the receiver acknowledges receipt of the message (e.g., Binary Synchronous Communications (IBM)), Digital Equipment's product line for Digital Network Architecture (Wecker,1980; Tanenbaum, 1981, pp.172-174), IBM's product line for Systems Network Architecture framework(Green, 1979; Atkins, 1980), the Defense Advance Research Project Agency Transport Control Protocol (Tanenbaum, 1981, pp.373-377) or CCITT's X.25 (Tanenbaum, 1981, pp.167-172). This mechanism would be used to transfer files from one computer to another, for example, over a local area network.

In our opinion, at the present time a great deal of guidance is required to engineer such systems (cf. the current literature: Bux, Kuemmerle, Truong, 1980; Easton, 1980; Fayolle, Gelenbe, Pujolle, 1978; Kleinrock, 1978A, 1978B; Reiser, 1979; Sunshine, 1976, 1977; Traynham, Steen, 1977; Yu, Majithia, 1979; Luderer, Che, Marshall, 1982; Luderer, Che, Haggerty, Kirslis, Marshall, 1981).

This work analyzes a model for a class of of protocols, so called *sliding window** protocols, for controlling the flow of information between a transmitter and receiver. Granted certain assumptions that are felt to be reasonable for local area network applications, it is possible to engineer a virtual circuit to achieve predictable performance. The protocol is described in detail elsewhere (Knuth, 1981; Tanenbaum, 1981, pp.148-164); here, in the interest of brevity, we will describe only the aspects pertinent to traffic handling characteristics. It is representative of a great many protocols currently in use, each of which differs in detail in terms of error handling, but not in terms of pacing the flow of data between the transmitter and receiver (cf Schwartz, 1982, for an analysis of a protocol with the same control of flow of data but a different acknowledgement strategy).

We ignore a wide variety of phenomena that must be dealt with in an actual system that in our opinion are sufficiently rare to have negligible impact on traffic handling characteristics. In particular, we ignore the impact of the local area network corrupting or losing packets, the impact of using the local area network and the interfaces for both control (virtual circuit set up and take down, packet acknowledgement) and data transfer, the impact of local area network delay being dependent upon the workload, and a variety of hardware and software failures.

Lest the reader feel that this problem is straightforward, we quote one authority:

> "*...flow control procedures are rather difficult to invent and extremely difficult to analyze...to date there is no satisfactory theory or procedure for designing flow control procedures, much less evaluating their performance...*" L.Kleinrock, 1978A

2. Summary

We determine quantitative tradeoffs between the time the transmitter and receiver spend on packet processing, local area network propagation delays, receiver storage space for buffering packets, and flow control parameters (in particular the maximum number of unacknowledged packets at the transmitter, denoted by W for window) to achieve different levels of performance (e.g., mean throughput rate and delay statistics).

A mathematical model or abstraction of an actual communication system is developed. First, we summarize the performance of this model using mean values for transmitter and receiver packet processing times before analyzing the impact of fluctuations on performance. In our opinion, the mean value analysis is probably the most important level of measurement and analysis in practice.

A system can be engineered to have a low message throughput rate and small message delays, or it can have a high message throughput rate and large message delays. The problem is to find the point at which message delay becomes unacceptable. We do so in two steps: first we find the maximum mean throughput rate, which presumably will lead to large delays; then we back off from the maximum mean throughput rate to a lower rate which will lead to smaller (acceptable) delays. There are three potential *bottlenecks* in this communication system, the transmitter, the receiver, and the local area network. In many applications, it is often desirable for the transmitter and receiver to be bottlenecks, not the local area network: put differently, we demand the different devices connected to the local area network be incapable of generating enough messages to generate unacceptable delays in message transmission through the channel.

* The term *window* arises from picturing a window onto the stream of packets at the transmitter, with the window open onto packets that have been transmitted but not yet acknowledged. The window *slides* as packets are acknowledged along with the transmitted packets.

From discussions with a variety of knowledgeable engineers, currently available products achieve a packet delay in the local area network of one hundred to five hundred microseconds, while packet transmitter and receiver processing delays of one to five milliseconds are typical. For this case, where network delays between the transmitter and receiver are negligible compared to packet processing times, when the transmitter and receiver are mismatched in speed by a ratio of two or more, i.e., either the transmitter is much faster or slower than the receiver, one or the other is a bottleneck, and the mean value analysis appears adequate for sizing mean throughput rate and mean delay. It is only in the intermediate region of comparable transmitter and receiver speeds that there is significant interaction between the choice of flow control parameters and message arrival and transmission time statistics.

Upper and lower bounds on mean throughput rate are determined using only the mean times to handle each stage of packet processing and the receiver buffer size; determining the lower bound was previously an unsolved problem. Moreover these bounds are *sharp*, in the sense that they can be achieved if the fluctuations about the means become sufficiently small or sufficiently large. Put differently, given *only* mean value information, these bounds quantify the *best* possible mean throughput rate. The method of analysis is based upon a systematic application of Little's Law (Little, 1961). The analysis leading to the upper bound is well known; the lower bound is evidently novel. The lower bound is *positive* and hence we have shown that at this level of analysis this protocol cannot *deadlock*. This occurs when the transmitter and receiver both have packets and acknowledgements to send but cannot because receiver buffer space is not available for the transmitter and vice versa.

Three cases are examined in detail: buffering one, two, and an infinite number of message packets at the receiver. Buffering one packet allows for *no* concurrency or parallel operation of the transmitter and receiver; buffering more than one packet allows for *some* concurrency and also *smooths* out bursts of packets, storing them until they can be processed. Performance (mean throughput rate and packet delay), can be significantly increased (provided the transmitter and receiver packet processing times are comparable to one another) in going from one to two packet buffering at the receiver, and is only marginally increased in going from two to infinite packet buffering. This is clear on physical grounds: there is only one transmitter and one receiver, and the best possible concurrency is achieved when both are busy, i.e., when the transmitter is filling one buffer and the receiver is emptying another. A single packet buffer at the receiver is a *bottleneck* limiting the maximum mean throughput rate of packet transmission; moving to two or more buffers at the receiver allows the *transmitter* or the *receiver* processor to become the bottleneck. Since it might not be known in practice which was the bottleneck, this suggests choosing two buffers in the receiver.

To proceed further, we make additional assumptions beyond mean values, to test both our modeling *assumptions* as well as the numerical *parameters*. In particular, we can quantify the impact of *fluctuations* about the mean values on performance.

The first case is a Jackson network model with associated product form distribution for the number of packets at each node in the network. The Jackson network assumptions allow us to obtain an *exact* calculation of mean throughput rate. The analysis uses techniques that are standard for Jackson networks (Kelly, 1976, 1979). In practice, packet processing times at the transmitter and receiver are widely felt to be much smaller than that found in this type of model, so this would lead to *pessimistic* performance. In fact, there is little difference between the mean throughput rate *upper bound* which is achieved with *no* fluctuations in packet processing times and the Jackson network analysis. In either case, choosing two buffers at the receiver achieves virtually all of the traffic handling gain.

The second model deals with the mean throughput rate and delay statistics with negligible network delay, for the special cases of W=1, W=2, and W=∞. If the transmitter and receiver packet processing times are independent identically distributed constant random variables, we obtain not simply mean values, but *distributions*. We show that the long term time averaged delay statistics for $W=2$ and $W \rightarrow \infty$ are *identical*. This was previously only conjecture.

3. System and Model Description

The system that motivated this work is described elsewhere (Fraser, 1979; Chesson, 1979, 1980; Luderer, Che, Haggerty, Kirslis, Marshall, 1981; Luderer, Che, Marshall, 1982).

3.1 Hardware Configuration

A set of terminals and computer systems are interconnected via a *local area network* switching system.

3.2 Functional Operation

A pair of digital systems communicate with one another as follows: After a full duplex virtual circuit is set up over the local area network, one system transmits a message to the other; the transmitter breaks each message over each virtual circuit up into packets, and stores the transmitted packets until the receiver sends an acknowledgement that transmitted packets were properly received.

When the system is started, the transmitter starts a packet sequence counter, denoted C, at zero. Messages are transmitted in order of arrival; packets within messages are transmitted in order over the virtual circuit. Each packet has a sequence number that is used to pace the flow of data from the source to the sink. Each time the transmitter sends a packet, C is incremented by one; each time the transmitter receives an acknowledgement, C is decremented by one and flushes this packet from its transmitter buffer. Hence, the packet sequence counter *slides* from the first packet to the last, with packets from different messages possibly interleaved. The maximum number of packets that can be buffered by the receiver is called the *window* denoted W. The largest the transmitter packet sequence counter can be is W: the transmitter knows the receiver can buffer at most this many packets. Each packet holds one receiver buffer; when the packet sequence counter strikes W the transmitter ceases to send messages, until a minimum number of acknowledgements are received. A start/stop protocol would have a window of size one (W=1): the transmitter would send the first packet of a message and wait for a positive acknowledgement before sending the next packet, and so forth. A double buffering protocol would have a window size of two (W=2).

How frequently should the receiver acknowledge packets? Ideally it should be done after each packet; however, if this requires an unacceptable amount of processor time at the transmitter or at the receiver or both, then acknowledgements could be batched. The normal operating regime is where control packets are much less frequent than data packets; hence, to first order, we might focus attention on the data packet mean processing time alone.

3.3 Queueing Network Model

The queueing model of this system (Figure 1) follows the above description quite closely.

3.3.1 Queues and Servers The system consists of a staging queue (with no server), a transmitter queue (with one server), a transmitter to receiver queue (with W servers), a receiver queue (with one server), and a receiver to transmitter queue (with W servers).

3.3.2 Packet Flow Through Queueing Network A packet migrates from one queue to another: packets arrive at an external queue where they are staged, before migrating to the transmitter queue, then through the transmitter to receiver queue, then to the receiver queue, then finally to the receiver to transmitter queue, before leaving the system; a packet is in the system if it is in the transmitter or receiver queue (waiting or in execution), in the propagation queue from the transmitter to the receiver and vice versa.

3.3.3 Service Required for Each Step of Packet Communications Each packet requires some processing time by the transmitter, denoted T_{trans}, including both packet processing time and data transmission time. Each packet propagates from the transmitter to receiver, in a mean time denoted $T_{trans-rec}$. Each packet requires receiver processing time, denoted T_{rec}. Each receiver acknowledgement packet propagates from the receiver to the transmitter in a time interval denoted $T_{rec-trans}$. The receiver and transmitter processing times are assumed to include the time to handle acknowledgement processing.

3.3.4 Flow Control Policy Arriving packets are stored in the staging queue. If there are less than W packets in the system, the packet immediately enters the transmitter queue; otherwise, the packet waits in the staging queue.

3.3.5 Phenomena Ignored By Model If a packet is not received by the receiver (e.g., because it was lost in transmission, because the receiver buffer overflowed, because the sink acknowledgement that the packet was received is lost, or for other reasons) the sender will not receive an acknowledgement within a given time interval called a *time out* interval (measured from the end of a given packet transmission) and the sender will retransmit the packet. If the receiver did in fact correctly receive a packet, when a new copy of that packet arrives it will be rejected and another acknowledgement will be sent. We ignore the impact of time outs, failures of different sorts, and noise which can garble a packet. Furthermore, we assume the packet delay due to the local area network is not a function of the message load, i.e., we assume the local area network simply adds delay to packet transmission. In a well designed system, these would be rare events, having little impact on performance.

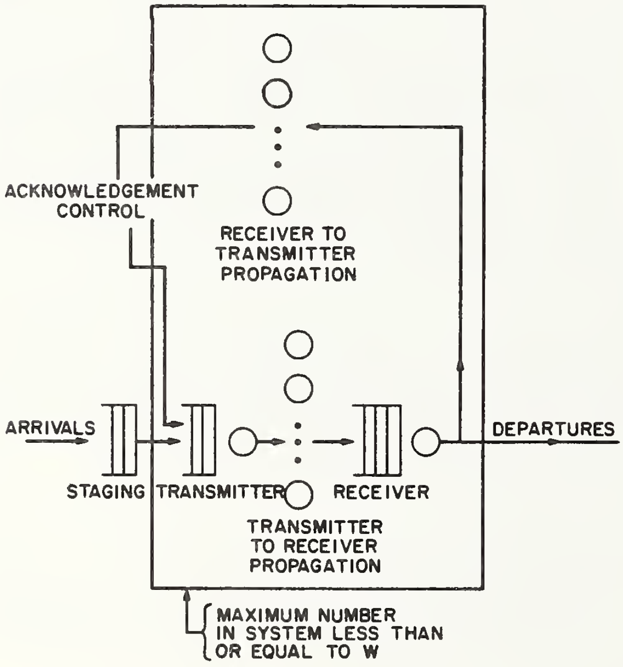

ACKNOWLEDGEMENT CONTROL
RECEIVER TO TRANSMITTER PROPAGATION
ARRIVALS
DEPARTURES
STAGING
TRANSMITTER
RECEIVER
TRANSMITTER TO RECEIVER PROPAGATION
MAXIMUM NUMBER IN SYSTEM LESS THAN OR EQUAL TO W

Figure 1.Queueing Network Block Diagram

4. Mean Throughput Rate Mean Value Analysis (cf Reiser, 1979; Fayolle et al, 1978)

The ingredients in the mean value analysis are

[1] The transmitter processes a packet; this step has a mean duration T_{trans} and it requires the transmitter processor

[2] The packet propagates over the link from the transmitter to the receiver; this step has a mean duration $T_{trans-rec}$.

[3] The receiver processes the packet; this step has a mean duration T_{rec} and it requires the receiver processor.

[4] Acknowledgements of correct receipt of the packet are batched up and then propagate from the receiver to the transmitter; this step has a mean duration $T_{rec-trans}$.

The acknowledgement processing per message at the transmitter and receiver is assumed to be included in T_{trans} and T_{rec} respectively.

In an appendix, we obtain upper and lower bounds on the maximum mean throughput rate, assuming there are alwavs sufficiently many packets at the staging queue that there are W packets in the system. Here we simply summarize the results:

$$\lambda_{lower} \leqslant \lambda_{\max} \leqslant \lambda_{upper} \tag{1}$$

$$\lambda_{upper} = \min \left[\frac{1}{T_{trans}}, \frac{1}{T_{rec}}, \frac{W}{T_{trans} + T_{trans-rec} + T_{rec} + T_{rec-trans}} \right]$$

$$\lambda_{lower} = \frac{W}{W[T_{trans} + T_{rec}] + T_{trans-rec} + T_{rec-trans}} > 0$$

The physical interpretation of the upper bound on mean throughput rate is as follows

- If the transmitter is the bottleneck, then

$$\lambda_{upper} = \frac{1}{T_{trans}} \tag{2a}$$

- If the receiver is the bottleneck, then

$$\lambda_{upper} = \frac{1}{T_{rec}} \tag{2b}$$

- If the number of buffers is the bottleneck, then

$$\lambda_{upper} = \frac{W}{T_{trans} + T_{trans-rec} + T_{rec} + T_{rec-trans}} \tag{2c}$$

The physical interpretation of the lower bound is that at most one packet at a time is being handled by the system. The practical import of the lower bound is that the maximum mean throughput rate is always *positive,* and hence the system cannot *deadlock* or stop transmitting packets.

4.1 Negligible Local Area Network Delay

In a local area network, packet delay due to the network is presumed to be negligible compared to packet delay at the transmitter and receiver. This special case merits closer examination. From this point on in this section, we assume $T_{trans-rec}=T_{rec-trans}=0$.

First, we assume that $W=1$. If we do so, we see

$$\lambda_{upper} = \min \left[\frac{1}{T_{trans}+T_{rec}} \right] \quad W=1 \tag{3}$$

In words, the maximum rate of transmitting packets is the reciprocal of the sum of the mean time spent by the transmitter plus the mean time spent by the receiver.

Increasing the number of buffers from one to two, $W=1$ to $W=2$ always increases the maximum mean throughput rate, and now we see

$$\lambda_{upper} = \min \left[\frac{1}{T_{trans}}, \frac{1}{T_{rec}} \right] \quad W>1 \tag{4}$$

Furthermore, this increase is maximized for $T_{trans}=T_{rec}$, and then the upper bound *doubles* in going from one buffer to more than one buffer. Why is this so? By having more than one buffer, both the transmitter and receiver can simultaneously be filling and emptying a buffer, allowing greater concurrency or parallelism compared with the single buffer case. We also note that allowing more than two buffers, e.g., *infinite* buffers, will not increase the upper bound on the maximum mean throughput rate any further; intuitively, we only have two serially reusable resources, the transmitter and receiver, and double buffering keeps them both busy simultaneously. This suggests investigating the three cases of single buffering, double buffering, and more than double buffering, in the subsequent sections. For the lower bound on mean throughput rate, we see that

$$\lambda_{lower} = \frac{1}{T_{trans} + T_{rec}} \qquad (5)$$

which is identical to the upper bound for W=1. Why is this so? There may be significant fluctuation about the mean values shown above, and in the limit of one big swing about the mean value all of the messages will pile up at one stage in the network and nothing will be transmitted until buffers become available.

4.2 Nonnegligible Local Area Network Delay

For the case where the local area delay is not negligible compared to the transmitter and receiver processing time per packet, the upper bound on mean throughput rate will increase as a linear function of the amount of buffering available at the receiver, until either the transmitter or the receiver becomes a limiting bottleneck. When does this in fact occur in our model?

Figure 2 plots these upper and lower bounds, as well as the results of an Jackson queueing network analysis (e.g., Kelly, 1976, 1979), for the special case where

$$T_{trans} = T_{rec} \qquad T_{trans-rec} = T_{rec-trans} \qquad (6)$$

In Figure 2A we have chosen the typical case, negligible network delay versus transmitter and receiver processing time, while in 2B all these times are equal to one another, and in Figure 2C the network delay is ten times the transmitter and receiver processing times.

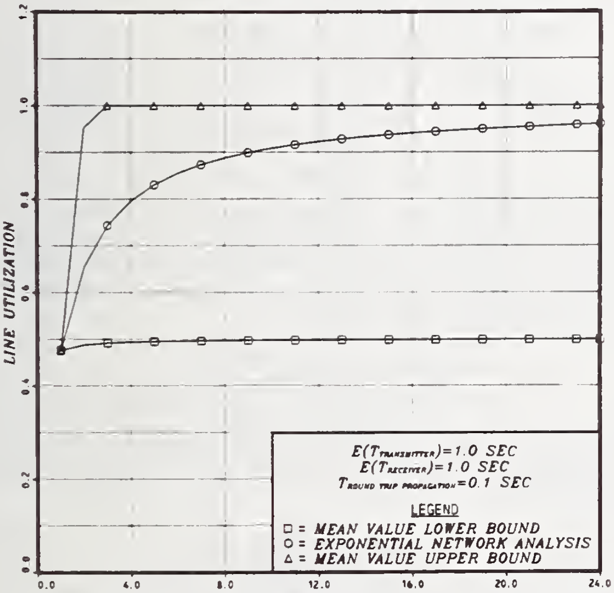


Figure 2A.Maximum Mean Throughput Rate vs *W*

The fraction of time the queueing network model predicts the system to be in state $\underline{J}$ is denoted by $\pi(\underline{J})$ where

$$\pi(\underline{J}) = \frac{1}{G} T_{trans}{}^{J_{trans}} \frac{T_{trans-rec}{}^{J_{trans-rec}}}{J_{trans-rec}!} T_{rec}{}^{J_{rec}} \frac{T_{rec-trans}{}^{J_{rec-trans}}}{J_{rec-trans}!} \qquad (7)$$

The system partition function denoted G is chosen to normalize the probability distribution:

$$\sum_{\underline{J}} \pi(\underline{J}) = 1 \qquad (8)$$

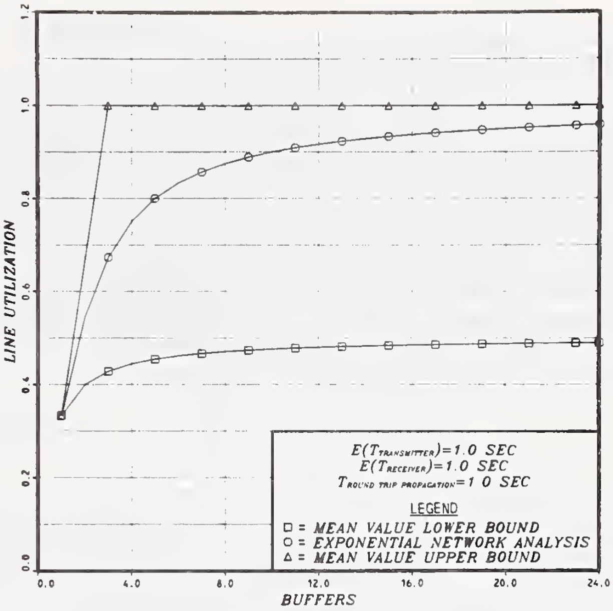


Figure 2B.Maximum Mean Throughput Rate vs *W*

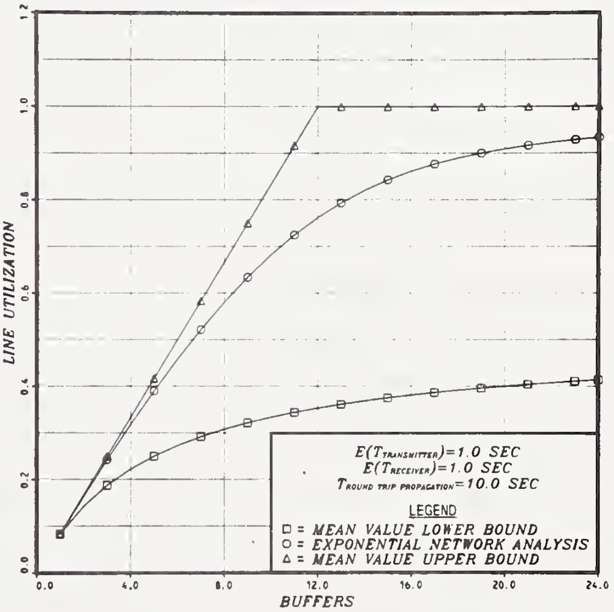


Figure 2C.Maximum Mean Throughput Rate vs *W*

Two regimes are evident, one where the buffers are the bottleneck and the mean throughput rate grows linearly in the number of buffers, and one where the receiver is the bottleneck and the mean throughput rate is independent of the number of buffers. For Figure 2A this occurs at W=2; for Figure 2B this occurs at W=4; for Figure 2C this occurs at W=22. As we see, there is little need for large buffers at the receiver in a local area network, at this level of analysis, provided the network delay is negligible compared to the packet processing delay. As is evident from the figures, the Jackson network analysis tracks quite closely the mean value upper bound on throughput rate.. Since the Jackson network analysis assumes the packet processing times have significantly greater fluctuation about their mean than in actual systems *constant* processing time per packet, and since the agreement (at this level of analysis) between the mean value upper bound and the Jackson network is quite close, this suggests using the mean value upper bound as a guide to setting flow control parameters, because it is quite straightforward to analyze.

5. Delay Statistics Analysis for W = 1 and W = 2 with Negligible Network Delay

For the special case of $W=1$ and $W=2$ with $T_{trans-rec}=T_{rec-trans}=0$, we wish to calculate the moment generating function of the packet delay random variable T_{delay}, which is measured from the instant a packet arrives until it is completely processed by the transmitter and receiver. We assume that packets arrive at the transmitter according to simple Poisson statistics, i.e., the packet interarrival times are independent identically distributed exponential random variables with mean interarrival time $1/\lambda$. The packet processing times at the transmitter and receiver are assumed to be independent identically distributed random variables denoted by T_{trans} and T_{rec}; these have associated moment generating functions

$$E[exp(-zT_{trans})] = \hat{G}_{trans}(z) \qquad (9)$$

$$E[exp(-zT_{rec})] = \hat{G}_{rec}(z) \qquad (10)$$

5.1 Delay Statistics for W = 1 with Negligible Network Delay

For $W=1$ the system acts as a single serially reusable resource whose service time is the sum of the transmitter and receiver packet processing times. The maximum mean throughput rate is the reciprocal of the sum of the mean packet processing times at the transmitter and receiver:

$$\lambda_{\max} = \frac{1}{E[T_{trans} + T_{rec}]} \qquad (11)$$

The moment generating function for packet delay is given by

$$E[exp(-zT_{delay})] = \qquad (12)$$
$$\frac{z(1 - \lambda E(T_{trans} + T_{rec}))}{z - \lambda[1 - G_{trans}(z)G_{rec}(z)]} \hat{G}_{trans}(z)\ \hat{G}_{rec}(z)$$

Packet delay is denoted by the random variable T_{delay}, measured from the time a packet arrives until it leaves, and its mean is given by

$$E[T_{delay}] = \frac{\lambda E[(T_{trans} + T_{rec})^2]}{2[1 - \lambda E[T_{trans} + T_{rec}]]} + E(T_{trans}) + E(T_{rec}) \qquad (13)$$

5.2 Delay Statistics for W = 2 with Negligible Network Delay

Figure 3 shows an illustrative arrival pattern and completion pattern of packets for the $W=2$ case. This suggests analyzing the delay statistics for this case in two stages:

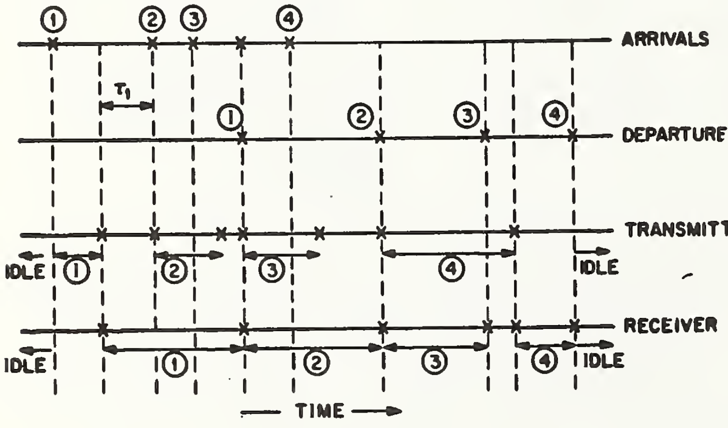


Figure 3.Illustrative Operation for $W=2$

[1] The first stage is the time from packet arrival at the transmitter to the start of service at the receiver

[2] The second stage is the time from the start of service at the receiver to the completion of service

Figure 4 shows a queueing network block diagram of this modified system.

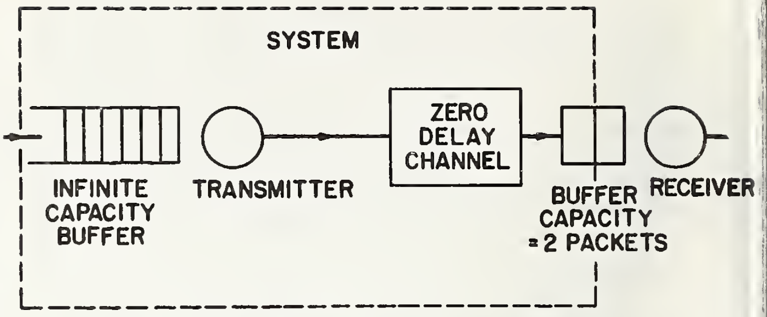


Figure 4.Queueing Network Block Diagram for $W=2$

The first stage acts as a modified M/G/1 queueing system, with the modification being that the initial service time of a busy period has a different distribution from the other service times during a busy period. The random variable T_{init} denotes the initial service time at the first stage during a busy period, and is given by

$$T_{init} = max[T_{trans}, T_{rec} - T_A] = T_{trans} + max[0, T_{rec} - T_A] \qquad (14)$$

where T_A is the arrival time of packet that ends the idle period and makes the transmitter busy. Note that for the case where $T_{trans} > T_{rec}$, e.g., when both times are constant, this initial interval involves transmitter delay, and all the delay will occur at the transmitter and none at the receiver. The random variable $T_{\max}$ denotes the service times of all but the first packet during a busy period:

$$T_{\max} = max[T_{trans}, T_{rec}] \qquad (15)$$

The moment generating functions for these random variables are denoted by

$$E[exp(-zT_{init})] = \hat{G}_{init}(z) \qquad (16)$$

$$E[exp(-zT_{\max})] = \hat{G}_{\max}(z) \qquad (17)$$

The distribution for the interarrival time is given by

$$PROB[T_A \leqslant X] = 1 - exp(-\lambda X) \quad \lambda = 1/E[T_A] \qquad (18)$$

If the packet delay distribution is nontrivial, the maximum mean throughput rate must be less than the mean of $T_{\max}$:

$$\lambda E[T_{\max}] = \lambda E[max(T_{trans}, T_{rec})] \leqslant 1 \qquad (19)$$

To determine the delay statistics, we look at completion times of packets finishing the first stage of processing. At this epochs we can associate an imbedded Markov chain, where $N_k, K \geqslant 0$ denotes the number of packets in system at the kth completion epoch. From earlier definitions, we can write

$$E[X^{N_{k+1}} | N_k = 0] = \hat{G}_{init}[\lambda(1 - X)] \qquad (20)$$

$$E[X^{N_{k+1}} | N_k = J > 0] = X^{J-1}\hat{G}_{\max}[\lambda(1-X)] \qquad (21)$$

The invariant measure associated with the imbedded Markov chain for $N_k, K \geqslant 0$ is denoted by π_k, which has moment generating function $\zeta(X)$:

$$\zeta(X) = \sum_{K=0}^{\infty} \pi_K X^K \qquad (22)$$

We now substitute into this expression using previously defined functions:

$$\zeta(X) = \pi_0 \hat{G}_{init}[\lambda(1 - X)] + \sum_{K=1}^{\infty} X^{K-1} \pi_K \hat{G}_{\max}[\lambda(1 - X)] \qquad (23)$$

Rearranging this, we find

$$\zeta(X) = \pi_0 \frac{X\hat{G}_{init}[\lambda(1 - X)] - \hat{G}_{\max}[\lambda(1 - X)]}{\hat{G}_{\max}(X) - X} \qquad (24)$$

In order for $\zeta(X)$ to be a proper moment generating function, we require $\zeta(1)=1$, and hence

$$\pi_0 = \frac{1 - \lambda E[T_{\max}]}{1 + \lambda E[T_{init} - T_{\max}]} \tag{25}$$

Now we have an expression for the moment generating function for the number in system at completion epochs to the first stage of packet processing. We want to relate this to packet delay. Because the arrival statistics were assumed to be Poisson, the moment generating function for the time spent in the first stage equals the moment generating function $\zeta(Y)$ evaluated at $Y=(\lambda - Y)/\lambda$, which is a deep generalization of Little's Law (Conway, Maxwell, Miller, 1967, EQ.20, pp.156). Hence, the total packet delay moment generating function is now known:

$$E[exp(-zT_{delay})] = \zeta\left[\frac{\lambda - z}{\lambda}\right] \hat{G}_{rec}(z) \tag{26}$$

$$= \frac{1 - \lambda E[T_{\max}]}{1 + \lambda E[T_{init} - T_{\max}]} \frac{(z - \lambda)\hat{G}_{init}(z) + \lambda \hat{G}_{\max}(z)}{z - \lambda[1 - \hat{G}_{\max}(z)]} \hat{G}_{rec}(z)$$

For the special case where the transmitter and receiver packet service times are constant, the delay *distribution* for W=2 is *identical* to that for $W \rightarrow \infty$.

The mean packet delay is given by

$$E[T_{delay}] = \tag{27}$$

$$\frac{E[T_{init}] + \frac{1}{2}\lambda E[T_{init}^2 - T_{\max}^2]}{1 + \lambda E[T_{init} - T_{\max}]} + \frac{\lambda E[T_{\max}^2]}{2[1 - \lambda E(T_{\max})]} + E(T_{rec})$$

The first term is due to the start of busy period and includes the transmitter packet processing time, the second term is due to the waiting time in the buffer, and the third term is the receiver packet processing time. For the special case where the transmitter and receiver packet processing times are constant, we have plotted illustrative results in Figure 5; the mean delay for W=2 is identical to that for $W \rightarrow \infty$, and for normal operating regions is significantly smaller than for W=1.

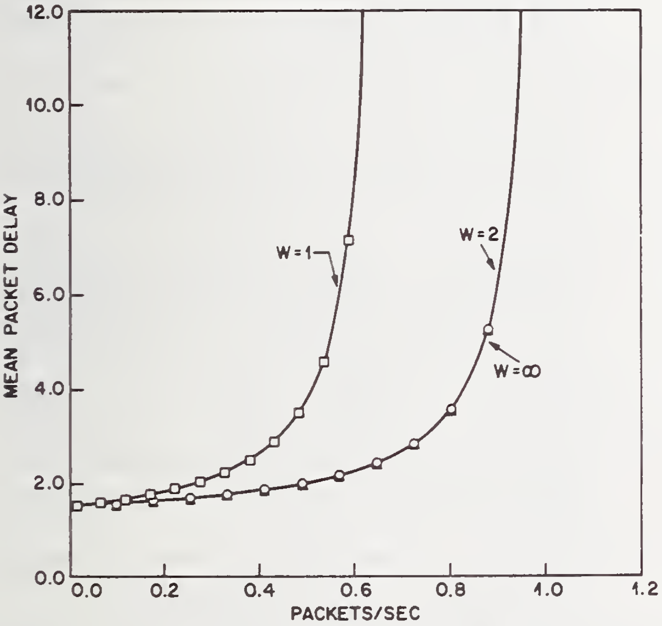


Figure 5.Mean Packet Delay vs Arrival Rate

expression for $W \rightarrow \infty$, which shows how close the W=2 delay statistics can in fact be.

6. Delay Statistics Analysis with Negligible Network Packet Delay for $W=\infty$

Our goal is to calculate the moment generating function of the packet delay random variable T_{delay}, measured from the instant a packet arrives at the transmitter until it departs the receiver. We examine two special cases for $W=\infty$ assuming $T_{trans-rec}=T_{rec-trans}=0$. The packet arrival statistics are Poisson: the interarrival times are independent identically distributed exponential random variables with mean interarrival time $1/\lambda$. The transmitter and receiver packet processing times are independent identically distributed random variables, with mean transmitter and receiver packet processing times denoted by T_{trans} and T_{rec} respectively.

6.1 Exponentially Distributed Packet Processing Times

The first case is where the transmitter and receiver packet processing times are mutually independent exponential distributions. This is a Jackson network (Kelly, 1976, 1979) and we merely cite the results

$$E[T_{delay}] = \frac{T_{trans}}{1 - \lambda T_{trans}} + \frac{T_{rec}}{1 - \lambda T_{rec}} \tag{28}$$

$$= \frac{\lambda T_{trans}{}^2}{2(1 - \lambda T_{trans})} + \frac{\lambda T_{rec}{}^2}{2(1 - \lambda T_{rec})} + T_{trans} + T_{rec}$$

The latter form is given to suggest that the mean packet processing times at the transmitter and receiver are augmented by a form suggestive of the mean waiting time for an M/M/1 queueing system, for the transmitter and the receiver.

6.2 Deterministic or Constant Packet Processing Times

The second case is where the transmitter and receiver packet processing times are deterministic or constant. Packets are assumed to arrive at time instants denoted by $t=0, A_1, A_1 + A_2, \ldots$ so that A_k is the interarrival time between the (k-1)th and kth packet. The time spent by the kth packet waiting to begin service at the transmitter and at the receiver is denoted by T_{trans,W_k} and T_{rec,W_k}, respectively.

For $W=\infty$, the following recursions specify the waiting time sequences for packets at the transmitter and receiver:

$$T_{trans,W_{k+1}} = max(0, T_{trans,W_k} + T_{trans} - A_{k+1}) \tag{29}$$

$$T_{rec,W_{k+1}} = max(0, T_{rec,W_k} + T_{rec} - (C_{k+1} - C_k)) \tag{30}$$

where C_k is the completion time of the kth packet by the transmitter:

$$C_k = A_1 + \cdots + A_k + T_{trans,W_k} + T_{trans} \tag{31}$$

Our goal is to show that the total time spent by a packet waiting at the transmitter and the receiver, denoted by T_{W_k}, is given by

$$T_{W_{k+1}} = max[0, T_{W_k} + T_{\max} - A_{k+1}] \tag{32}$$

where $T_{\max}=max[T_{trans}, T_{rec}]$ is the larger of the two packet processing times.

Two cases arise. First, if the transmitter is the slower of the pair, i.e.,

$$T_{trans} = T_{\max} = max[T_{trans}, T_{rec}] \tag{33}$$

then $T_{rec,W_k}=0$ for all values of k, i.e., there is no waiting at the receiver at all. The closest spacing in time of packets departing from the transmitter is greater than T_{rec}, and hence we have shown

$$T_{W_{k+1}} = max[0, T_{W_k} + T_{\max} - A_{k+1}] \tag{34}$$

The second case, $T_{trans} < T_{\max}$, can be handled in two steps. First, if

$$T_{trans,W_k} + T_{trans} - A_{k+1} \geqslant 0 \tag{35}$$

then from the recursions we see

$$C_{k+1} - C_k = T_{trans} \tag{36}$$

and hence

$$T_{trans,W_{k+1}} + T_{rec,W_{k+1}} \tag{37}$$

$$= T_{trans,W_k} + T_{trans} - A_{k+1} + max[0, T_{rec,W_k} + T_{\max} - T_{trans}]$$

$$= T_{trans,W_k} + T_{rec,W_k} + T_{\max} - A_{k+1} \geqslant 0$$

or in other words

$$T_{W_{k+1}} = T_{W_k} + T_{\max} - A_{k+1} \tag{38}$$

as was desired. The second special case is the converse: if

$$T_{trans,W_{k+1}} = 0 \quad T_{trans,W_k} + T_{trans} - A_{k+1} < 0 \tag{39}$$

so that

$$C_{k+1} - C_k = A_{k+1} - T_{trans,W_k} + T_{trans,W_{k+1}} \tag{40}$$

Substituting this into the earlier recursions, we see

$$T_{trans,W_{k+1}} + T_{rec,W_{k+1}} \tag{41}$$

$$= max[0, T_{trans,W_k} + T_{rec,W_k} + T_{\max} - A_{k+1}]$$

or in other words

$$T_{W_{k+1}} = max[0, T_{W_k} + T_{\max} - A_{k+1}] \tag{42}$$

and hence we have obtained our desired result.

Using this recursion, the mean packet delay is given by

$$E[T_{delay}] = T_{trans} + T_{rec} + \frac{\lambda T_{\max}{}^2}{2(1 - \lambda T_{\max})} \quad \lambda T_{\max} < 1 \tag{43}$$

6.3 Interpretation of Results

For the first case, the transmitter and receiver processing times exponentially distributed, the mean packet delay *is* identical to that of two M/M/1 queueing systems. The mean packet processing time is inflated or multiplied by the reciprocal of the fraction of time each stage is idle.

For the second case, the transmitter and receiver processing times deterministic or constant, the mean packet delay *looks* like that of a single M/D/1 queueing system, with the slowest stage contributing all the waiting time, while all the stages contribute to the packet processing time delay. For the case where $T_{trans} > T_{rec}$ this will be the case; for the case where $T_{trans} < T_{rec}$, in fact there will be *some* delay at the transmitter stage, simply due to the bursty nature of the Poisson process (i.e., an arrival has a nonzero probability of occuring within the transmitter packet processing time of a previous arrival), but the *system* behaves as if all the waiting were at the transmitter. Due caution is needed.

References

[1] E.Arthurs, B.W.Stuck, *Upper and Lower Bounds on Mean Throughput Rate and Mean Delay in Queueing Networks with Memory Constraints,* Bell System Technical Journal, **62** (2), 541-581 (1983).

[2] J.Atkins, *Path Control: The Transport Network of SNA,* IEEE Transactions on Communications, **28** (4) 527-538 (1980).

[3] W.Bux, K.Kuemmerle, H.Linh Truong, *Balanced HDLC Procedures: A Performance Analysis,* IEEE Trans. Communications, **28** (11), 1889-1898 (1980).

[4] G.L.Chesson, *DATAKIT Software Architecture,* 20.2.1-20.2.5, Proceedings International Communications Conference, Boston, Massachusetts, 1979.

[5] G.L.Chesson, A.G.Fraser, *DATAKIT Network Architecture,* COMPCON'80, 25-28 February 1980, San Francisco, California, IEEE Catalog Number 1491-0/80, pp.59-61

[6] R.W.Conway, W.L.Maxwell, L.W.Miller, **Theory of Scheduling,** Addison-Wesley, Reading, Massachusetts, 1967; Little's formula, pp.18-19.

[7] M.C.Easton, *Batch Throughput Efficiency of ADCCP/HDLC/SDLC Selective Reject Protocols,* IEEE Trans.Communications, **28** (2), 187-195 (1980).

[8] G.Fayolle, E.Gelenbe, G.Pujolle, *An Analytic Evaluation of the Performance of the "Send and Wait" Protocol,* IEEE Trans.Communications, **26** (3), 313-319(1978).

[9] A.G.Fraser, *DATAKIT--A Modular Network for Synchronous and Asynchronous Traffic,* 20.1.1-20.1.3, Proceedings International Communications Conference, Boston, Mass., 1979.

[10] P.E.Green, *An Introduction to Network Architecture,* IBM Systems Journal, **18** (2), 202-222 (1979).

[11] G.C.Hunt, *Sequential Arrays of Waiting Lines,* Operations Research, **4**, 674-683 (1956).

[12] IBM, *General Information--Binary Synchronous Communications,* **IBM Systems Reference Library,** GA27-3004-2.

[13] IBM, *MSP/7 Binary Synchronous Communications: Program Logic Manual,* GY34-0012-1.

[14] F.P.Kelly, *Networks of Queues,* Advances in Applied Probability, **8**, 416-432 (1976).

[15] F.P.Kelly, **Reversibility and Stochastic Networks,** Wiley, Chichester, 1979.

[16] L.Kleinrock, *Principles and Lessons in Packet Communications,* Proceedings IEEE, **66** (11), 1320-1329(1978;A).

[17] L.Kleinrock, *On Flow Control,* Proc. Int. Conf. Communications, Toronto, Canada, 27.2-1--27.2-5, June 1978;B.

[18] D.E.Knuth, *Verification of Link-Level Protocols,* BIT, **21**, 31-36 (1981).

[19] Wolfgang Kramer, *Investigations of Systems with Queues in Series,* Institute of Switching and Data Technics, University of Stuttgart, Report #22(1975).

[20] J.D.C.Little, *Proof of the Queueing Formula L=λW,* Operations Research, **9** (3), 383-386(1961).

[21] G.W.R.Luderer, H.Che, W.T.Marshall, *A Virtual Circuit Switch as the Basis for Distributed Systems,* Journal of Telecommunication Networks, **1**, 147-160 (1982).

[22] G.W.R.Luderer H.Che, J.P.Haggerty, P.A.Kirslis, W.T.Marshall, *A Distributed UNIX System Based on a Virtual Circuit Switch,* ACM Operating Systems Review, **15** (5), 160-168 (8th Symposium on Operating Systems Principles, Asilomar, 14-16 December 1981), ACM 534810.

[23] M.F.Neuts, *Two Queues in Series With a Finite, Intermediate Waiting Room,* J.App.Prob., **5**, 123-142 (1968).

[24] M.F.Neuts, **Matrix Geometric Solutions in Stochastic Models,** Section 5.2, pp.217-232, Johns Hopkins Press, Baltimore, MD, 1981.

[25] M.Reiser, *A Queueing Network Analysis of Computer Networks with Window Flow Control,* IEEE Trans.Communications, **27** (8), 1199-1209(1979).

[26] M.Schwartz, *Performance Analysis of the SNA Virtual Route Pacing Control,* IEEE Transactions on Communications, **30** (1), 172-184 (1982)

[27] C.A.Sunshine, *Factors in Interprocess Communication Protocol Efficiency for Computer Networks,* AFIP NCC, pp.571-576(1976).

[28] C.A.Sunshine, *Efficiency of Interprocess Communication Protocols for Computer Networks,* IEEE Transactions on Communications, **25**, 287-293(1977).

[29] A.S.Tanenbaum, **Computer Networks,** especially Section 4.2, pp.148-164, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[30] K.C.Traynham, R.F.Steen, *SDLC and BSC on Satellite Links: A Performance Comparison,* Computer Communication Review, 1977

[31] S.Wecker, *DNA: The Digital Network Architecture,* IEEE Transactions on Communications, **28** (4), 510-525 (1980).

[32] L.W.Yu, J.C.Majithia, *An Analysis of One Direction of Window Mechanism,* IEEE Trans.Communications, **27** (5), 778-788(1979).

Modelling and Monitoring a LAN,
One Experience

W. Bruce Watson

Lawrence Livermore National Laboratory
Livermore, California

This is a partial summary of efforts to design, model, measure and optimize a large, Hyperchannel-based, local network. I elaborate upon the cyclic interaction of these four activities and contrast our successes and failures in each with its costs.

Keywords: discrete event simulation; Hyperchannel based network; model validation; network monitoring; network performance evaluation.

1. Introduction

Clark, Pogran and Reed[1] have suggested that the issues of local network design can be classified as either configuration issues or protocol issues. They visualize networks as consisting of four basic elements; the transmission medium, a control mechanism, the interfaces and the protocols. Network performance is not only strongly dependent upon each of these elements, but also upon their mutual interactions.

As has been pointed out by Sunshine[2], Pouzin and Zimmermann[3], and others, network traffic properties such as message sizes, rates and their distributions have a great effect on network performance, and further that performance is not only strongly dependent on the traffic but also upon the mutual interaction of the traffic, the configuration and the protocols.

The study and characterization of these interactions is one of the goals of current local network research at LLNL More specifically, the Local Network Research Group (LNRG) is currently investigating the traffic characteristics and network performance in an operational High Data Rate Local Network (HDRLN) as part of a study being conducted for the National Bureau of Standards (NBS). As currently envisioned, the NBS sponsored investigation will not only rely on monitoring and measuring techniques, but also make extensive use of a *model*, a comprehensive, discrete event, computer, simulation of Network Systems Corporation's (NSC) Hyperchannel

Monitors and models can be and have been of critical importance in the

Work funded by the National Bureau of Standards and performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under the contract number W-7405-ENG-48.

design, implementation and tuning of local area networks (LAN) and network operating systems (NOS). For example, modelling and simulation can verify and explore the anticipated performance of tentative LAN designs well in advance of their actual implementation. Once implemented, modelling and monitoring can help the designer better understand and tune the implementation enabling him to enhance its performance in some desirable way. This is basically an iterative or feedback process that is a part of the larger, cyclic, design-measure-model process depicted in figure 1. Modelling and monitoring provide the feedback without which effective design is not possible. This design-measure-model process is certainly nothing new and I only mention it here to stress its cyclic nature and that much of its usefulness derives from the fact that it is cyclic. Modelling, for example, is a way of checking out a design through simulating its behavior in advance of (or concurrently with) its actual implementation. Ideally, as flaws are detected or enhancements discovered during the design's simulation, beneficial modifications are fed back into the original design and its model, the whole process being repeated. This is a cyclic process that is hopefully speedily and cheeply convergent. Today, no serious designer of network protocols and hardware fails to simulate his designs. It is probably the only practical way of knowing whether complex state diagrams are complete and accurate, or of selecting one from among a set of competing designs.

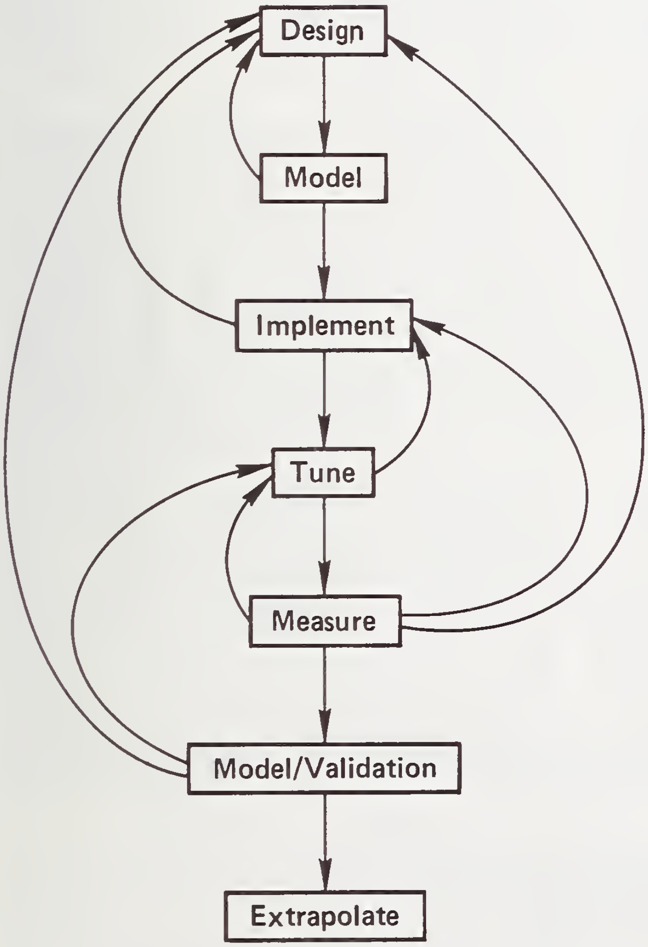


Figure 1. The cyclic, design-model-measure process (↓), showing various possible feedback paths (↑).

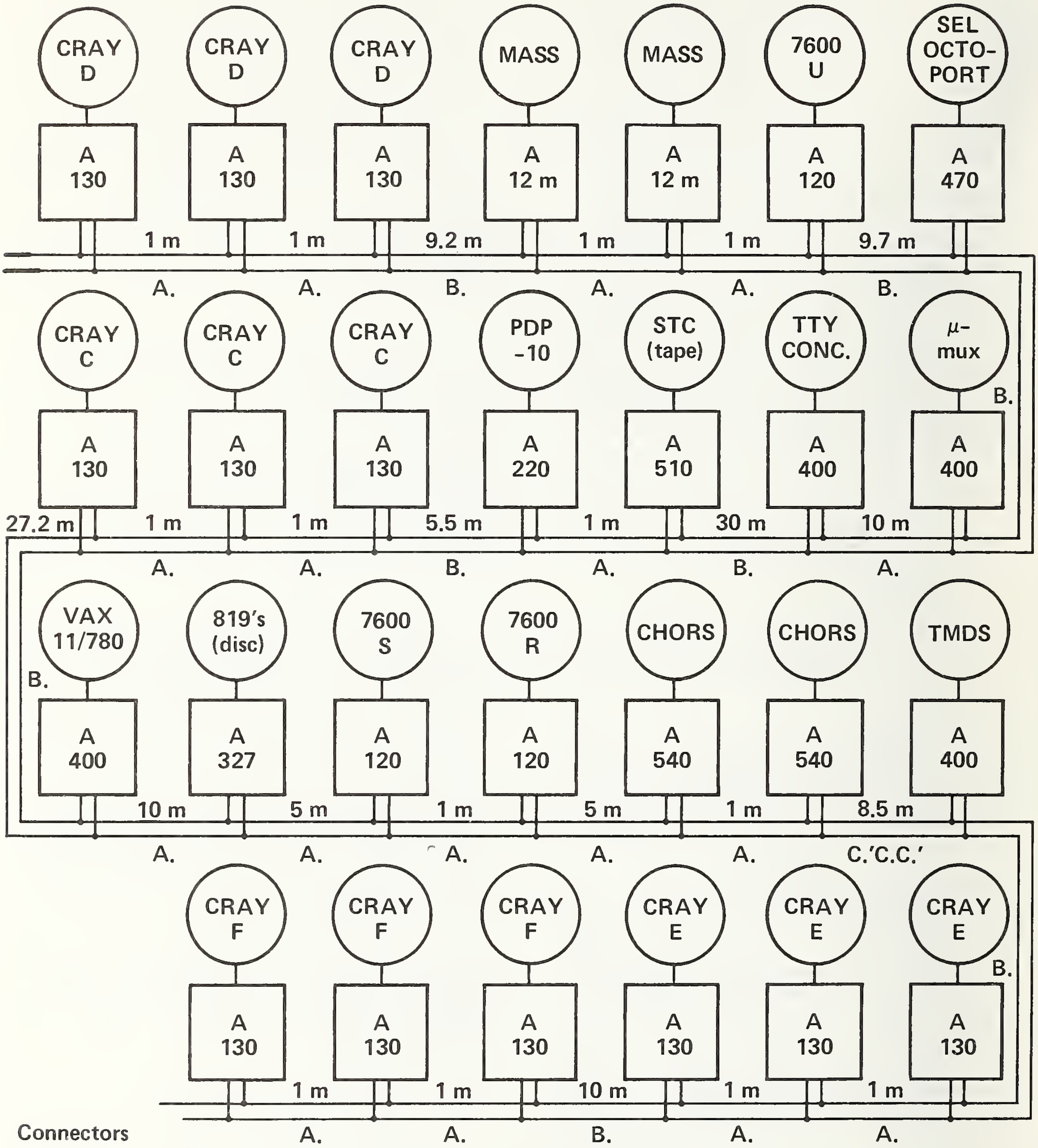
Figure 1 indicates that the points of feedback in this cyclic process occur at the modelling and measuring locations. Without modelling and measuring, successful and efficient design and implementation of things as complex as networks becomes difficult if not altogether impossible. If you don't (can't) model it, how can you hope to understand it, and if you don't measure it how do you know if it's working right (how can you hope to fix it)?

So, the NBS project will proceed by a composite of simulation and measurement methodologies in the following way: we will use a monitor to study the performance of LLNL's HDRLN, the Craynet (a Hyperchannel based LAN); we will also use this monitor to validate LLNL's discrete event, computer simulation of the Hyperchannel; and we will use this validated simulation to characterize the performance of a HDRLN at extrapolated loads (loads derived from measured values by extrapolation).

I shall use the cyclic, design-model-measure paradigm as the schema throughout the remainder of the paper for elaborating upon our experiences, our successes and failures, and the costs.

2. Design

The Craynet, figures 2, 3 and 4, a high data rate local network (HDRLN), is the only access the user's have to LLNL's complement of three (soon to be four) Cray computers. It is also the only access the Crays have to facilities such as the long term storage and transfer of



Connectors

A. – LEA 76-903301 (RG59B/U)
KC59-232 (BNC)

B. – BELDEN 9248 (RG 6/U)
TEI 14949-UPL20-41

C. – ANDREWS FHJ4-75
ANDREWS 44AN75 (N-Series)

C.' – BELDEN 9292

Figure 2. The CRAYNET, a schematic showing connectors, spacing, adapter types and attached hosts.

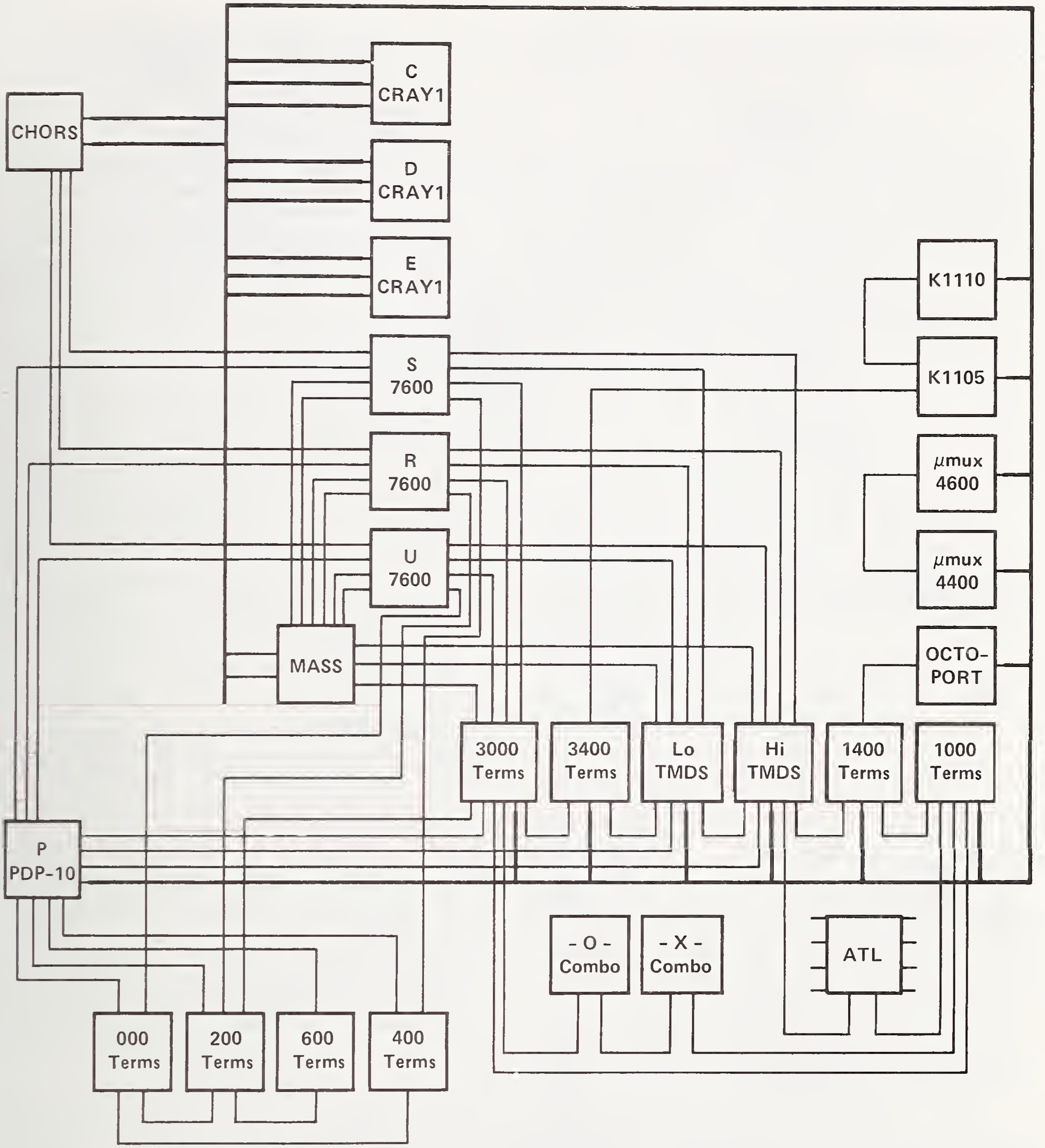


Figure 3. OCTOPUS NETWORK, with the CRAYNET subnet shown in heavy black lines.

files as well as to the usual output devices.

Figure 2 is a schematic diagram of the various Network Systems Corporation (NSC) Hyperchannel adapters that are attached (or will soon be attached) to the Craynet. It shows their spacing along the Hyperchannel coaxial cable, the connectors each one uses as well as the computer/service to which each one is connected.

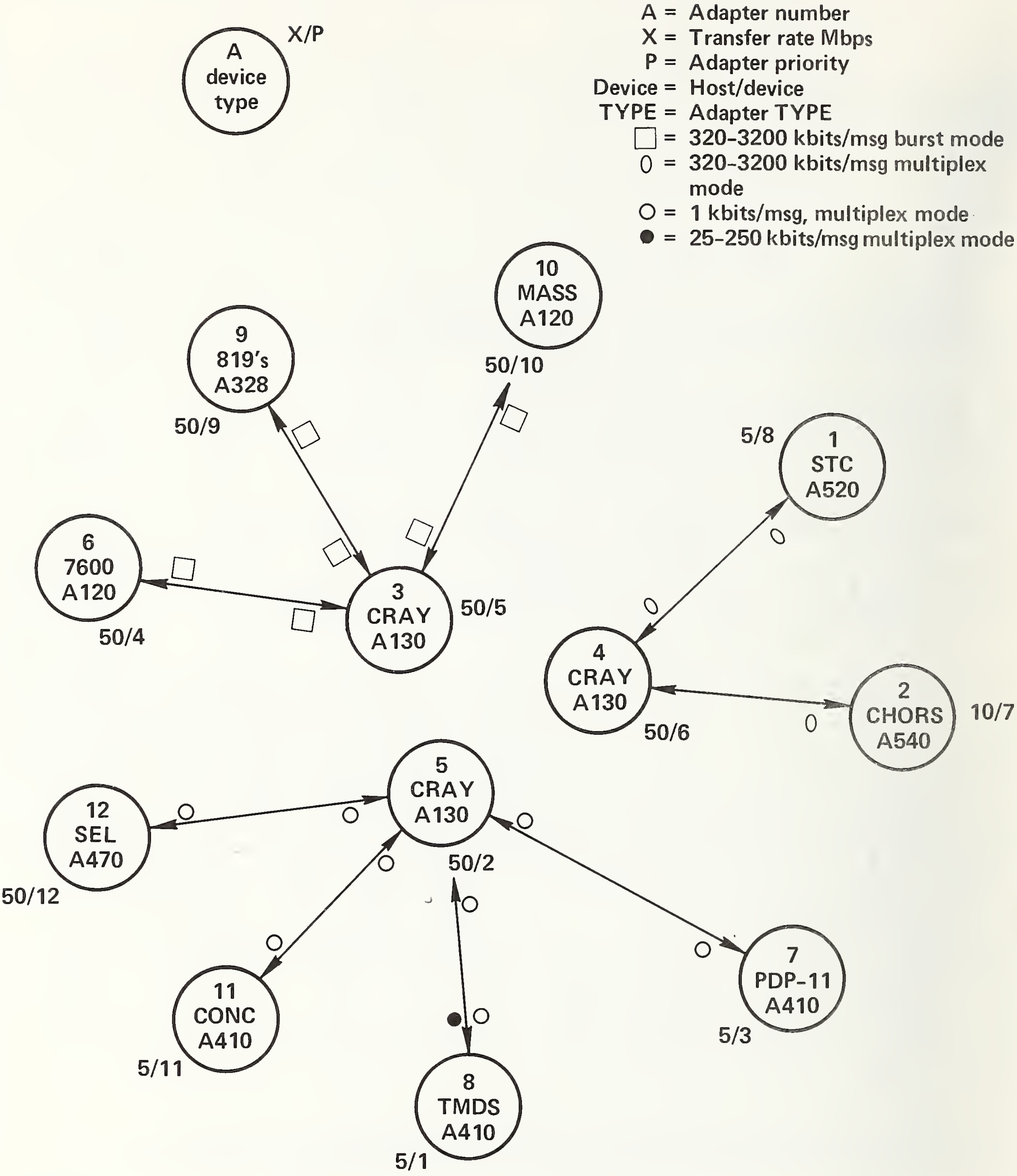


Figure 4. An older, single CRAY version of CRAYNET showing traffic patterns and message sizes.

Octopus, figure 3, LLNL's computer network, has evolved over the last 15 years as a packet switched, store and forward, partially connected mesh network. The Hyperchannel based Craynet is represented in figure 3 by the heavy black lines. It is the latest addition to the Octopus network.

Figure 3 depicts the logical interconnection of the three Cray computers, a plethora of storage devices, output devices and terminal concentrators. The Craynet, a broadcast network, is thought of as an adjunct to the Octopus network. It was looked upon as a simple way of achieving many point to point paths amongst all the various host machines, output/storage devices, and terminal concentrators for all newly acquired computers.

Indeed, some of the traffic on the Craynet is store and forward traffic from the Octopus network—internetwork traffic requiring certain machines to act as gateways. For example, files from the Crays destined for the automated tape library (ATL) must pass through the multiple access storage system (MASS), which acts in this instance as a gateway to the older part of the octopus network. The actual path of transfer is then, Cray⇒MASS⇒7600-S⇒PDP-10⇒ATL. I mention this because we at LLNL tend to think of and use (and possibly short sightedly so) the Hyperchannel as just another link in the existing message network, the Octopus.

Table 1 presents the data paths within the Craynet indicating the message size and modes of transmission per path. It is interesting but not surprising to observe that many paths are not used. Indeed, if one considers only the major traffic and assumes that messages never need to travel in store and forward mode, the Craynet configuration logically begins to resemble a star, with the three Cray adapters as the central node and the concentrator/storage/output adapters as the peripheral nodes. Figure 4 depicts this for an older, single Cray version of the Craynet.

The Crays are large, fast computers operated under timesharing. Their principle use is in the numerical solution of large systems of partial differential equations pertaining to physics calculations. During the day, most programmers/physicists are developing new versions of their calculations or debugging existing ones. In addition, a large number of users are preparing input to these calculations to be performed during nightly production runs. Finally, a lot of users use these machines to further process and analyze results and output from previous nights' production runs. During the night and on weekends, the Crays are mainly used in production mode.

There are radical differences between the daytime, user generated tasks and nighttime, production oriented tasks of the Crays. During the day, the tasks are numerous, small and execute for short periods of time. The opposite conditions occur at night. Because the timesharing system is tuned to reflect this disparity, interactivity is very poor during the nighttime hours, and large programs seldom get loaded or executed during the daytime hours.

3. Model

It is unfortunate from the standpoint of this cyclic paradigm whose virtues I am trying to extol that the Hyperchannel, the Craynet, and the Octopus network itself were designed and implemented without the aid of any formal modelling or computer simulation techniques.

Who knows how much easier it would have been to do all of this, or how much more efficient these designs could have been had modelling been used? Unfortunately, history does not record its alternatives, and it's difficult if not silly to try to argue with success.

As an aside, it's worth noting that during the initial design of Ethernet, extensive use was made of modelling and computer simulation techniques to eliminate the useless or deleterious features, and optimize the favorable ones. By such a process, its designers finally arrived at what has come to be called Ethernet—carrier sense multiple access with collision detection (CSMA/CD) with binary backoff and retry. Both the original Ethernet and the now commercially available product perform excellently the tasks for which they were designed.

The Hyperchannel, on the other hand, which predates the Ethernet by quite a few years as a commercial product, was designed and implemented almost entirely without the use of models and simulation. With its CSMA media access method and link level protocols, it is capable of operating faster than 8000 messages a second (512 bytes per message), i.e. 120

Table 1. Showing message sizes, paths and transmission modes for the CRAYNET. Here, ○ represents 1.2-250 kbits/msg sent in multiplex mode; ● represents 0.49 mbits/msg sent in burst mode; and 0 represents 0.49 mbits/msg sent in multiplex mode.

To From	Adapter #	CRAY -C	-C	-C	CRAY -D	-D	-D	CRAY -E	-E	-E	CRAY -F	-F	-F	MASS -1	-2	SEL	μmux	PDP -11	STC	PDP -10	VAX	819 's	TMDS	7600 -R	7600 -S	7600 -U	CHORS -1	CHORS -2
Adapter #		12	13	14	1	2	3	22	23	24	25	26	27	4	5	7	8	9	10	11	15	16	21	18	17	6	19	20
CRAY-C	12	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	●	●							●		☒		☒		
-C	13	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒						0					☒		☒	0	0
-C	14	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒			○	○	○		○	○		○	☒		☒		
CRAY-D	1	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	●	●							●		☒		☒		
-D	2	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒						0					☒		☒	0	0
-D	3	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒			○	○	○		○	○		○	☒		☒		
CRAY-E	22	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	●	●							●		☒		☒		
-E	23	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒						0					☒		☒	0	0
-E	24	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒			○	○	○		○	○		○	☒		☒		
CRAY-F	25	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	●	●							●		☒		☒		
-F	26	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒						0					☒		☒	0	0
-F	27	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒			○	○	○		○	○		○	☒		☒		
MASS-1	4	●			●			●			●			☒	☒	○		○		○	○	●	○	☒	●	☒		
-2	5	●			●			●			●			☒	☒	○		○		○	○	●	○	☒	●	☒		
SEL	7			○			○			○			○			☒	○	○		○	○	●	○	☒		☒		
μ-mux	8			○			○			○			○				☒	○		○	○		○	☒		☒		
PDP-11	9			○			○			○			○					☒		○	○		○	☒		☒		
STC	10		0			0			0			0							☒					☒		☒		
PDP-10	11			○			○			○			○							☒	○		○	☒	●	☒		
VAX	15			○			○			○			○								☒		○	☒		☒		
819's	16	●			●			●			●			●	●	●						☒		☒		☒		
TMDS	21			○			○			○			○	○	○	○	○	○		○	○		☒	☒		☒		
7600-R	18	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒
7600-S	17													●	●					●					☒			
7600-U	6	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒
CHORS-1	19	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒		☒	☒	☒
CHORS-2	20	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒		☒	☒	☒
Adapter Type		${}^{1}3_{0}$	${}^{1}3_{0}$	${}^{1}3_{0}$	${}^{1}3_{0}$	${}^{1}3_{0}$	${}^{1}3_{0}$	${}^{1}3_{0}$	${}^{1}3_{0}$	${}^{1}3_{0}$	${}^{1}3_{0}$	${}^{1}3_{0}$	${}^{1}3_{0}$	${}^{1}2_{m}$	${}^{1}2_{m}$	${}^{4}7_{0}$	${}^{4}0_{0}$	${}^{4}0_{0}$	${}^{5}1_{0}$	${}^{2}2_{0}$	${}^{4}0_{0}$	${}^{3}2_{7}$	${}^{4}0_{0}$	${}^{1}2_{0}$	${}^{1}2_{0}$	${}^{1}2_{0}$	${}^{5}4_{0}$	${}^{5}4_{0}$

micro-seconds per message. Yet when the various hosts in the Craynet, figures 2, 3 and 4, are subjected to loads such as given in tables 1, 2 and 3, they are unable to present to the network a load anywhere near this magnitude, perhaps a few hundred messages per second on a sustained basis (and during peaks?). Well, in 1977, as part of a research project at LLNL to evaluate Hyperchannel

Table 2. CRAYNET message sizes, frequencies, modes and paths where BM = data stream mode, and MM = packet mode.

Source Adapter # (= priority)	Sink Adapter #	Msg. Size (K-bits)	Max. Msg. Frequency (msg/sec)	Msg. Mode*	Description
1 (12,22,25)	4,5	490		BM	CRAY-D to MASS (file xfer)
1 (12,22,25)	16	490		BM	CRAY-D to CDC 819 DISKS
2 (13,23,26)	10	490	<.07(?)	MM	CRAY-D to STC tapes
3 (14,24,27)	7	2-250(?)		BM	CRAY-D to SEL
3 (14,24,27)	8,9	.2-.8	4	MM	CRAY-D to terminals
3 (14,24,27)	11,15	.2-.8	(?)	MM	Store & fwd backup terminal msgs
3 (14,24,27	16	32-262		MM	CRAY to TMDS (Text-Pix Display)
3 (13,23,26)	19,20	360	5	MM	CRAY to CHORS
4, 5	12,14,22,25	490		BM	MASS to CRAYs (file xfer)
4, 5	17	490		BM	MASS to 7600 (file xfer store/fwd)
4, 5	7,8,9,11,21	.018-.8	(?)	MM	Store & fwd backup terminal msg
6					
7	3,14,24,27	.018	≤ 2(?)	MM	SEL to CRAYs terminal MS
7	8,9,11,15	.018-.8	(?)	MM	SEL to TTYs (store & fwd), I/O
7	21	(?)	(?)	MM	SEL to TMDS (?)
8	3,14,24,27	.018	≤ 2(?)	MM	μ-mux to CRAYs input TTY msg
8	9,11,15,21	.018-.8	(?)	MM	μ-mux, store & fwd TTY msg, I/O
9	3,14,24,27	.018	≤ 2(?)	MM	PDP-11 to CRAYs input TTY msg
9	11,15,21	.018-.8	(?)	MM	PDP-11, store & fwd TTY msg I/O
10	13,2,23,26	490	(?)	MM	STC Tape to CRAYs
11	3,14,24,27	.018	(?)	MM	PDP-10 to CRAYs store/fwd TTY input
11	17	490	(?)	BM	PDP-10 to 7600-S store/fwd xfer
15	3,14,21,24,27	.018-.8	(?)	MM	VAX store/fwd TTY msg I/O
16	1,12,22,25	490	(?)	BM	819's to CRAYs file xfer
17	4,5	490	(?)	BM	MASS to CRAYs file xfer
17	11	490	(?)	BM	7600s to PDP-10 store/fwd file
18					
19					CHORS
20					CHORS
21	3,14,24,27	.018-.8	(?)	MM	TMDS to CRAY msg store/fwd
16	4,5,7	490	(?)	BM	SEL/MASS store/fwd file xfer via 819
21	4,5		(?)	MM	TMDS to MASS
21	7,8,9,11,15	.018-.8	(?)	MM	TMDS, store/fwd TTY msg I/O

Key to Tables 1 and 2

STC	STORAGE TECHNOLOGY CORP.	tape controller and 6250 bpi drive
CRAY	CRAY RESEARCH CORP. CRAY-1	computer
7600	CONTROL DATA CORP. 7600	computer
PDP-11	DIGITAL ELECTRONICS CORP.	secondary terminal concentrator
VAX	DIGITAL ELECTRONICS CORP.	computer
PDP-10	DIGITAL ELECTRONICS CORP.	computer
TMDS	LCC designed	television display monitor system
819's	CONTROL DATA CORP.	network disk(s)
SEL	SYSTEMS ENGINEERING CORP.	internet gateway (octoport)
μ-MUX	LCC designed	terminal concentrator 9600 baud
MASS	LCC designed	multiple access storage system (CDC 38/500, TI-980
CHORS	LCC designed	computer hardcopy output recording system (18000 lpm printer and μfilm)

based networks and their potential, a discrete event computer simulation of the Hyperchannel was developed. The initial thrust of this project was to explore the performance of such networks, but very quickly its goal became the detailed examination of Hyperchannel functionality (adapter hardware and protocols). Much was learned and much, reported,[4,5,6,7,8,9] mostly about certain shortcomings and inadequacies of these protocols, and their impact on performance at loads higher than occur in the Craynet.

I bring this up only to indicate the feedback that took place subsequently in the design of their new product, the Hyperbus. NSC did cooperate extensively with us during the development of the Hyperchannel model, and corrected those problems brought out by simulation that were ecomically feasible to correct. However, a few potentially serious shortcomings remained, notably in the area of buffer management. Today if one compares Hyperchannel functionality with that of the new Hyperbus, one can't help but observe that many of these shortcomings were corrected in its design.

What I am describing here is a design-model feedback path that transcended not only a succession of products, but also the relationship between vendor and customer, where all of the designing occurred in a private corporation and all of its simulation occurred in a government laboratory. As one could expect, such a feedback path is greatly attenuated by the conflict between a corporation's need to protect its ideas and the government's insistence that the results of all publicly funded research end up in the public domain. Add to this conflict of needs the legal restriction that the U. S. government may not fund a project that would result in an unfair advantage for some corporation(s) over its competitors.

As another example, I have been assured on several occasions by people within Control Data Corporation (CDC) that not only did they benefit from our studies of the Hyperchannel in the design of their own network product, the Loosely Coupled Network (LCN), they also made great use of their own computer models and simulations.

And lastly, the American National Standards Institute (ANSI) also benefitted from the LLNL and CDC simulations. That is, the ANSI committee, X3T9.5, proposed standard, the Local Distributed Data Interface (LDDI) that is currently in the final stages of approval, embodies basically a Hyperchannel-like design with, in my opinion, all of its shortcomings and inadequacies corrected. The point I am trying to make is that in at least three instances a design was improved by the feedback of information obtained as a direct result of the simulation of the original design.

What did all of this cost? The costs lie mainly in three categories: language costs, model development costs, and simulation execution costs. We used the ASPOL language (a CDC software product for 6600 and 7600 machines only), which operates only under CDC operating systems. As simulation languages go, it was fairly inexpensive to acquire and to master (compared to CACI's SIMSCRIPT), but not particularly powerful or ubiquitous (compared again to SIMSCRIPT). The model required about one half of a man year to program and debug. Finally, computer simulation used about an hour of CDC-7600 cpu time per study (depending of course on the nature of the study) for the type we did.[4,5,6,7,8,9]

Before I leave this section on modelling/simulation, I'd like to explain how the model was used to help design a network monitor (currently being built for LLNL by an outside corporation entirely with funds provided by NBS). If the reader wishes to avoid reading a lot of gory detail about buffer size determinations, he may skip ahead to the measurement section.

The Hyperchannel Monitor Device (HMD), figure 5, will detect, time-stamp, bus-label and write to magnetic tape selected portions of all frames that appear on each Hyperchannel transmission cable (bus) to which it is attached. Specifically, the selected portions are the frame header and part of the frame body, figure 6.

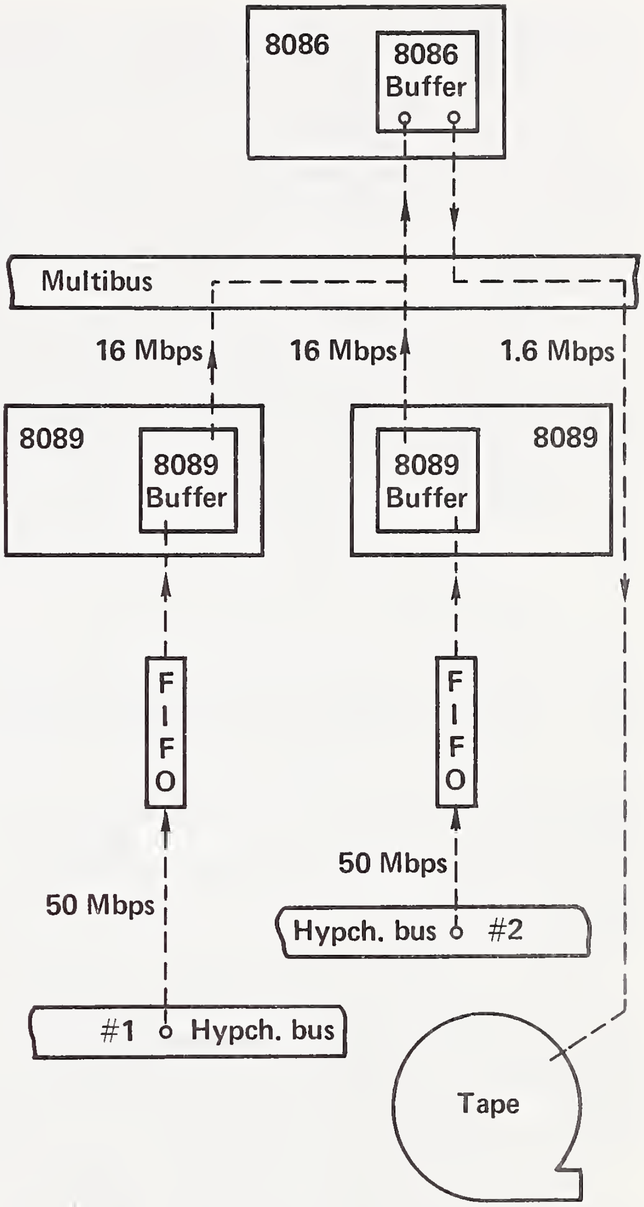


Figure 5. A schematic of the HMD showing data paths, (dotted lines), position of critical buffers and FIFOs, and maximum, data transfer rates in megabits/second.

The HMD is totally passive as far as the Craynet is concerned in that it never originates any network messages of its own, nor is ever the explicit recipient of any. It will not impair the normal operation of the Craynet in any way. It attaches to the Hyperchannel at the physical end of the buses replacing the terminators that are usually there.

Since each bus normally operates at a rate of 50 megabits/second (Mbps), the HMD must be able to copy bits from each bus into internal buffers at this same

Part	Field	Bits
Frame header	Sync	24
	Frame code	8
	Access code	16
	To	8
	From	8
	Response	16
	Length count	16
	Header checkword	16
Data field	Data	n
	Data checkword	16
	Sync	56

Fig. 6. Hyperchannel frame format.

rate. The 50 megabits/second rate is the instantaneous maximum rate at which bits could arrive at the HMD. However, due to the time multiplexed aspect of the Hyperchannel bus access strategy, and the fact that it is only the fronts of frames that are copied, the average, sustained maximum data rate is only a few megabits/second.

As the HMD collects data, it writes it to a magnetic tape. Data reduction and data base management routines will subsequently analyze the data on these tapes on some suitable computer(s) in the Livermore Computer Center (LCC).

At the outset, by far the most critical design question we had to answer was, Could the various internal buffers in the HMD keep up with the data rates expected to exist on the Craynet? To answer this question, we used the discrete event simulation model of the Hyperchannel. We subjected the modelled Hyperchannel, figure 2, to a steady state load similar to those discussed in the above section and depicted in the plots at the end of this memo. As far as we could tell, this load is real and typical, and amounts to about 5% of the Craynets capacity.

Briefly reviewing the internal buffers of the HMD, figure 5, there is a FIFO buffer that receives data from the Hyperchannel bus, i.e. there is one FIFO per bus in the network. Each FIFO is capable of receiving data at the bus rate of 50 Mbps. Each FIFO consists of 32 (or 64, 96, 128) 128-byte entries. Each entry can hold the data captured from one and only one Hyperchannel frame. Not all of the bits in a Hyperchannel frame are desired, only the leading bits containing information about the frames function, size and routing, figure 6. So, no more than 77-bytes of data are ever captured from any Hyperchannel frame. There is, in addition, a 5-byte time stamp and a 1-byte bus label that must ultimately be appended to these data bytes prior to their being written to tape. Consequently, the maximum amount of data to be written to tape for each frame appearing on a Hyperchannel bus is 83-bytes. To simplify the subsequent retrieval of data from this tape, all tape records must be some multiple of this 83-byte logical record.

To continue, the FIFO empties into an intermediate buffer in the 8089 bus board in which it resides. Each 8089 buffer is easily capable of keeping pace with its FIFO, (but not with the Hyperchannel bus), so this interface is not likely to be a bottleneck and requires no further attention.

Each 8089 buffer empties over the multibus into the 8086 buffer, moving over this path at a maximum rate of 16 Mbps. Finally, the 8086 buffer empties to tape, also over the multibus, at a maximum rate of 1.6 Mbps (1600BPI at 125IPS).

We will ignore for the time being such concerns as whether or not a FIFO can simultaneously fill from the bus side and empty out the 8089 side. Similarly, we need to know that the 8089 and 8086 buffers can empty and fill simultaneously, and if not, what effect this will have on their data transfer rates.

The first question examined with the aid of the simulation model was, Can a FIFO keep up with the Hyperchannel bus for the presented loads? Two 32X128-byte FIFOs can empty at 16 Mbps in about 4 milliseconds, assuming that all 128 bytes of each entry moves to the appropriate 8089 buffer regardless of whether they hold data or not. If this is the case, we next ask how many frames will appear on the two Hyperchannel buses in any given 4 millisecond window?

Specifically, is it more than 64? What we are dealing with now is not bit rates but frame and entry rates. The model indicates that for the given load, monitored for a 20 second period, 85% of the 4 millisecond windows contain fewer that 10 frames, but the probability is almost 100% that at least three 4 millisecond windows will contain in excess of 64 frames. The implication is that the FIFOs will overflow once every 6.66 seconds for the given load on the average. That is, no FIFO of any depth can hope to keep up with the Hyperchannel if it is emptied as described above.

On the other hand, if only data bits are transferred, we get a different result. In this case, we can think of a 128-byte-wide entry as merely an addressing convenience, and that when data moves from these entries to the 8089 buffer, only data, time stamp and bus-label bits are taken. When operated in this fashion, we can ignore the frame rates and concentrate simply on the bit rates associated with the data captured from the buses.

For the same load used in the above experiment, we observed: that the average number of bytes captured from the Hyperchannel frames was about 45 bytes/frame; that the peak data rate for both buses combined was 60 Mbps, for a 4 millisecond period; that the peak capture rate for both buses combined was about 8 Mbps for a 4 millisecond period; and that the average data rate for the captured data over the 20 seconds of the experiment was about 380 kbps.

Since the FIFOs are emptied at 16 Mbps, it does not appear that a two 32 entry FIFOs will ever overflow for the loads modelled even when we consider the extra load created by the addition of the 6 bytes of time stamp and bus label. That is, each captured, 45-byte record is expanded to 51 bytes, thus resulting in an apparent FIFO-to-8089 buffer bandwidth of about 45/51st's, or 84%, of its actual value.

How often do peaks occur on a Hyperchannel bus under a normal load, a load that we have tried to model given the data plotted in the figures within this report? We can not answer this question. This is one of the questions we hope to answer once we have the HMD. Unfortunately, it is one of the things we needed to know in order to properly design it. However, if we subject the simulated network to the steady state load described herein, *contention peaks* occur. A contention peak is one that occurs due to the randomizing effects of the contention algorithm upon bus access granted to various nodes with various size messages to send.

The model does give us a feeling for what the magnitude, frequency and duration of these contention peaks is. The two buses together peak at a rate of 60 Mbps for a duration of 4 milliseconds, once every 20 seconds. Lesser peaks occur more frequently. For example, a 40 Mbps peak of 4 milliseconds duration occurs once every 0.2 seconds.

The next questions we answer are, What was the average rate at which bits were captured during the simulated time? and How does this compare to the 8086-to-tape write rate?

During the 20 seconds of simulated operation, data was captured at a rate of 380 kbps. Again, the average number of captured bits/hyperchannel frame is 45 bytes. These captured data bits are augmented with 6 bytes of time stamp and bus label, expanding the number of bits destined for tape from 45 to 51 bytes worth. However, to maintain simplicity in the subsequent retrieval of data from this tape, these 45 (plus the 6 above) bytes flow onto tape in 83-byte records, (or multiples of 83-byte records). We recall that simplicity is often expensive as we note that imbedding 45 bytes of data in 83 byte records acts to diminish the effective bandwidth of the tape, i.e. a reduction from 1.6 Mbps to .867 Mbps. Fortunately for us, 867 kbps is more than adequate to keep up with the average capture rate of 380 kbps.

What we need to know next is the size of the 8086 buffer required to handle the peaks likely to occur. Again, since we know nothing about these peaks, the best we can do is explore the effects of contention peaks. For the sake of argument, lets postulate an 8086 buffer that requires one second to write to tape, realizing that the rate at which data is written to tape depends on a lot of factors such as multibus availability and the average number of bits captured/Hyperchannel frame. We simulated the load described herein, looking at .5 second windows trying to determine if any of these windows contained more captured bits than would fit into half of some size 8086 buffer. That is we are simultaneously trying to determine the size of the 8086 buffer and the effective 8086-to-tape transfer rate.

We found that during 205 seconds of simulated time, the average captured data rate was 412 kbps and that the average number of bytes capture/Hyperchannel frame was 45. The fact that this latter average is so high means that data could move to tape at more than half (45/83^{d}'s) its peak rate or at .867 Mbps. This implies that an 8086 buffer that is as large as or larger than .867megabits could keep up with the contention peaks associated with the steady state load simulated. Further examination of simulated results shows that of the 410 .5 second windows in this experiment, none contained more than 350 kbits.

The same experiment was conducted assuming an 8086 buffer half as large as above, i.e. 512 kbits, capable of being written to tape in .5 seconds. Looking at .25 second windows, we observed that about 1% of them contain more than 256 kbits, i.e. more than half of the 8086 buffer. This implies that 1% of the time a 512 kbits buffer would fail to buffer the input peaks.

One final observation is that as the number of bits per message presented to the Hyperchannel decreases, the average number of bytes captured per Hyperchannel frame decreases. The implication is clear; the effective 8086-to-tape bandwidth will decrease as a consequence. That is, the tape will be less likely to keep up for two reasons; data (mainly protocol frames) is arriving too quickly, and the transfer bandwidth is diminished.

I think I can conclude from all of this that for the loads modelled, any reasonably large 8086 buffer, e.g. 1024 kbits, should enable the tape to keep up with the Hyperchannel, assuming that the multibus can accommodate a doubly buffered 8086 buffer.

Those were the major design questions we had to answer before we could have any confidence that a monitor could be built, a monitor that would capture the desired data almost 100% of the time. It is possibly worth noting two things in passing. First, the above use of the model is a premier example of the cyclic design process depicted in figure 1. For I have described how the network model helped design the network monitor, the monitor which itself subsequently will help validate the model. And second, once there is a computer simulation, it is relatively easy, inexpensive, and informative to perform a peripheral study such as this one.

4. Measure

There are at least two ways to measure the performance and behavior of a broadcast network. One can install software network wide in every host operating system and thereby measure and monitor every host's network related activity. However, he would have to design such software uniquely for each different host system. Presumably the results of such monitoring would somehow be collected later at some central point for analysis and interpretation. Alternatively, one can simply just monitor the broadcast network's transmission medium.

Each of these two techniques has its advantages and disadvantages. For example, using host resident software, it is easy to determine network message queues and host to host transmission delays, and network throughput on a per host basis; but it is almost if not in fact totally impossible to synchronize the initialization of host monitoring software network wide so that all the measurement periods cover exactly the same period of real time. In addition, if the network itself is used for the continuous collection of measurement data at a central sight, then the network's performance will have been effected by the measurement process. Finally, if network monitoring software was not included in the design of each host's operating system at its outset, adding it later is usually prohibitively expensive if not impossible to accomplish given a diversity of hosts such as in the Craynet. On the other hand, using the second alternative, medium monitoring, network message queue length information is inaccessible, and transmission delay can only be inferred and then only during periods of low loads (no message queues). However, network throughput information is precisely knowable for arbitrarily small or large measurement periods. Ideally, one would proceed by using both techniques, and indeed we at LLNL are attempting to measure the Craynet by a composite of both methods.

4.1 Software monitor

What follows next is a brief description of the kinds of things we've been able to measure so far using monitoring software residing in some (but by no means all) hosts. Following that is a commentary upon and evaluation of our results so far.

The Craynet is currently monitored by the systems of some of the host computers connected to it. The data accumulated by these host systems is periodically and routinely collected, analyzed and plotted. Much useful information is thereby available concerning the use that the various hosts make of the network. The Cray operating system monitors all of its own traffic into and out of the Craynet. It is not practical nor desirable to present all of the results of the data thus collected. However, I shall present some of the more interesting results pertaining to peripheral I/O, terminal message and file transfer activity. The plots, figures 7 to 17, were obtained from just the Cray-C machine, the most heavily used of the four Cray computers.

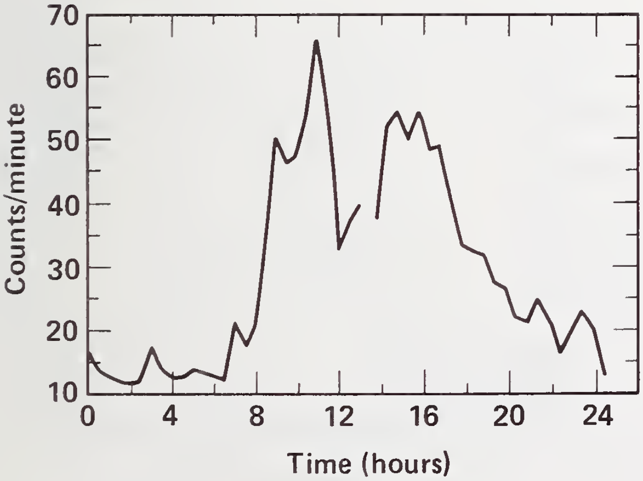


Figure 7. The number of first-time initializations of user programs per minute.

In figure 7 is plotted the number of user-job initializations/minute during a 24 hour period. The number of initializations peaks at a value of 65/minute just before lunch.

In figure 8 is plotted the number of user-jobs loaded into main memory for execution under timesharing per minute during a 24 hour period. The number of loads peaks at a value of 350/second just before lunch.

In figure 9 is plotted the number of output files generated per minute during a 24 hour period. The number peaks at a value of 4/minute at 7 pm. This peak occurs then, because a lot of low priority output files, which are

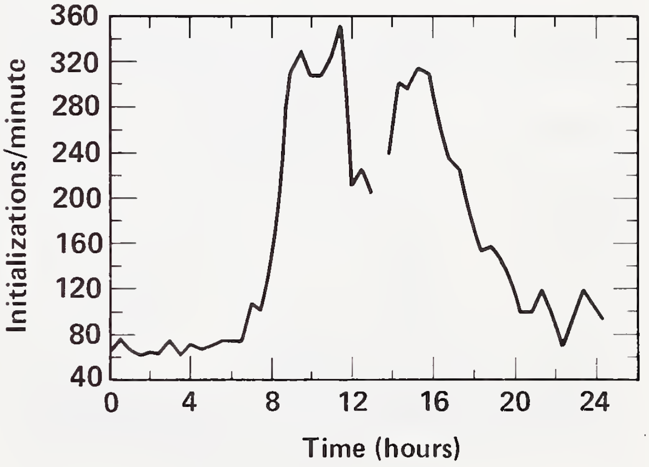


Figure 8. The total number of user programs initializations per minute.

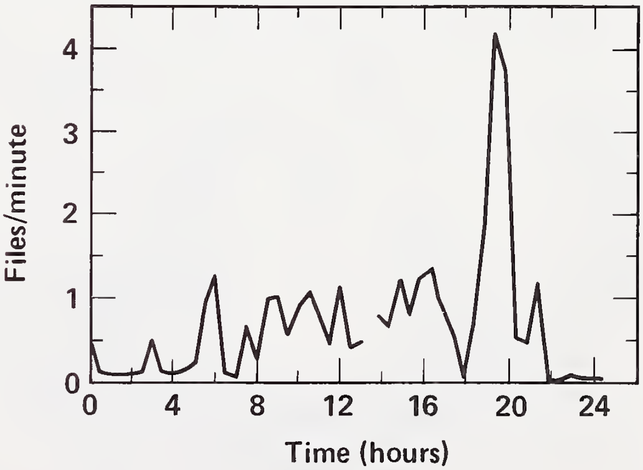


Figure 9. The total number of user/program generated output files per minute.

accumulated during the day, are off-loaded at night. Incidentally, some of these files are sent over the Cray-Chors path (see table 2 and figure 4).

Figures 10, 11 and 12 plot data that pertains to disc file creations and data transfers to/from the Cray machines disks. It is interesting to note, figure 12, the transfer rate reaches a peak value of 50 megabits/second, a peak that is maintained for hours at a time. None of this disc activity involves the Craynet or Hyperchannel.

The previous plots and discussion are intended to give the reader some feeling for the kinds and intensity of activities taking place on the Cray machines. It is one of the purposes of the NBS study to somehow relate this activity and intensity to the traffic measured on the Craynet. However, this work can not continue until the Hyperchannel monitor device becomes available.

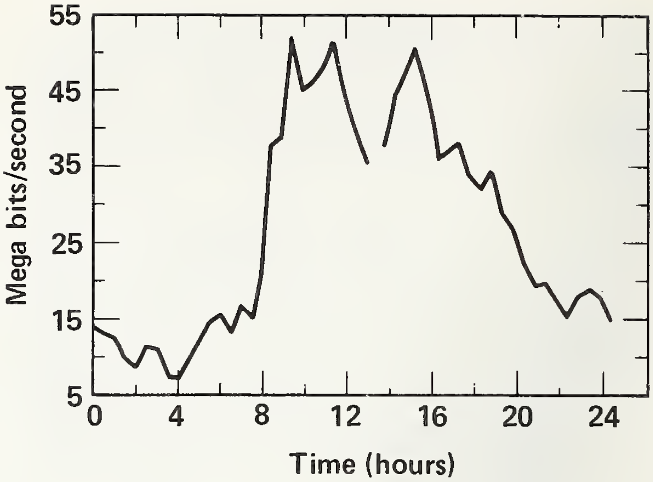


Figure 12. The total number of bits/second of disk traffic into/out of CRAY-C machine.

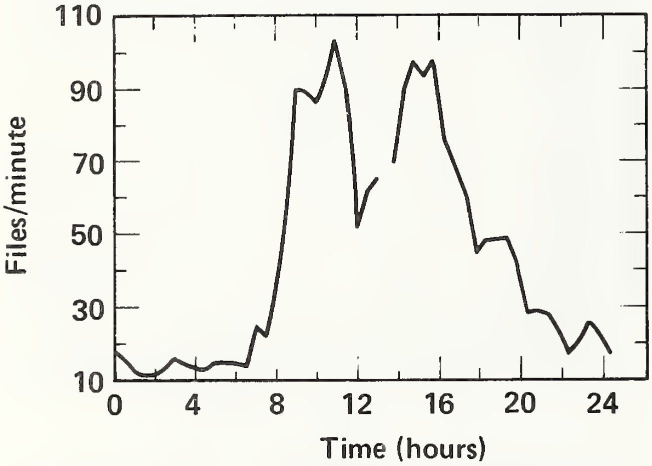


Figure 10. The total number of files created by user/utility programs per minute.

Figures 13 and 14 depict the terminal traffic in and out of the Cray C-machine over a 24 hour day. These figures show that the maximum input rate is about 350 messages/minute, (8000 bytes/minute). These messages arrive at the Cray-C machine over a variety of paths from the different terminal concentrators, (see table-1). It is not possible to distinguish amongst the

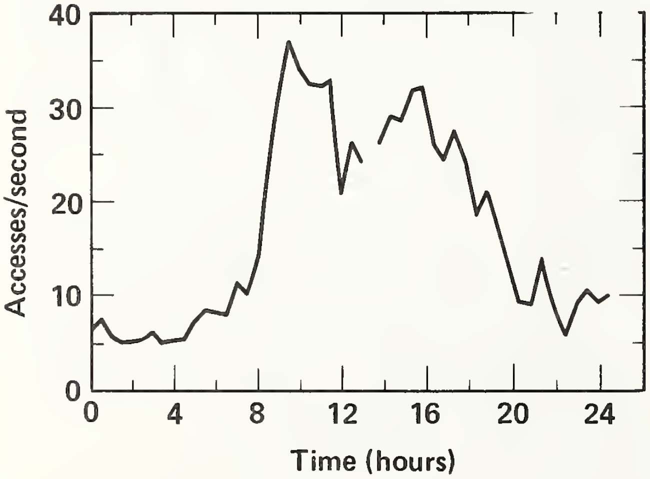


Figure 11. The total number of disc accesses per second caused by user/utility/system programs.

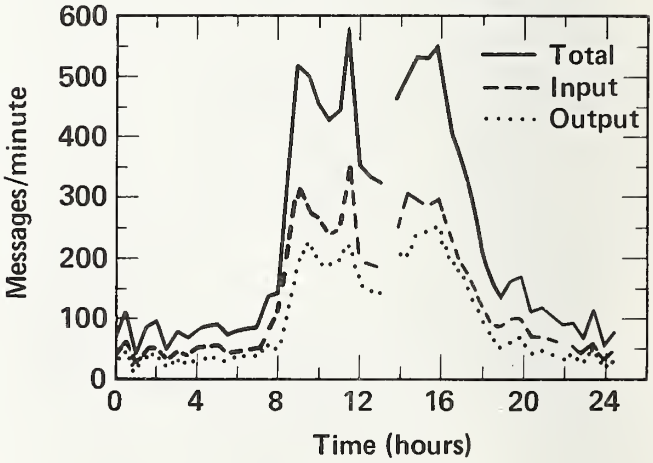


Figure 13. The total message traffic to/from CRAY-C machine for all terminal concentrators.

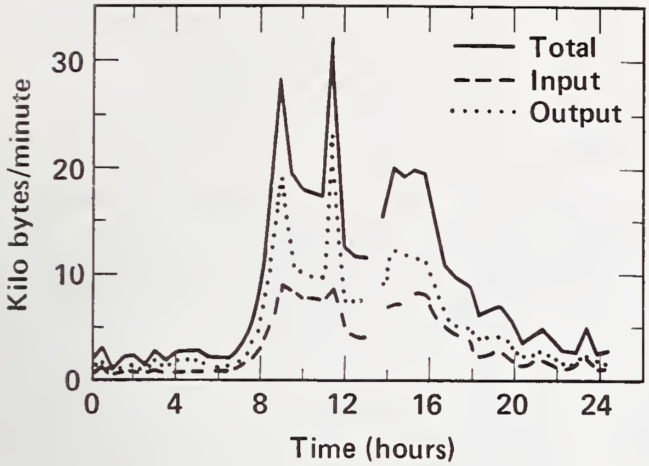


Figure 14. The total message traffic, in kilo bytes/ minute, into and out of the CRAY-C machine for all concentrators.

contributions of these various paths to this data.

The television monitor display system (TMDS) provides the users with various display facilities at their terminals. By far the most common use to which the TMDS is put is in the area of word processing. The display of textual data requires relatively small network messages, i.e 32 kilobits/message. The TMDS is also used for the display of graphical or plotted data usually generated during the nighttime production run of some physics calculation. The messages required to support such pictures contain about 256 kilobits each. In figure 15, we see displayed the rate of data flow over a 24 hour day from the Cray-C machine to the TMDS.

File transfers, in bits/second, to and from the MASS storage device are plotted in figure 16, and reach a peak value of 3.5 megabits/second around 3 pm. The maximum network message associated with such file transers contains around 200 kilobits.

The computer hardcopy output recording system (CHORS) provides the users with their principle output service. It is to the CHORS utility that files are sent when destined to become paper, film or fiche. Figure 17 shows the data traffic from the Cray-C machine to the CHORS utility adapter. This path peaks at a value of 210 kilobits/second.

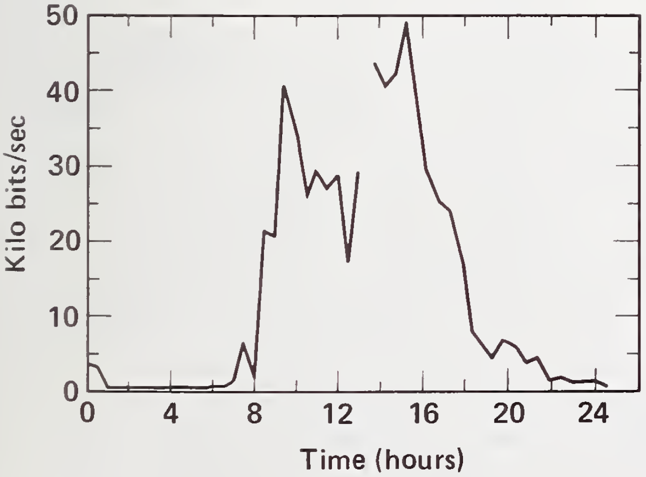


Figure 15. The total traffic, in kilo bits/second, of text and picture data into the TMDS.

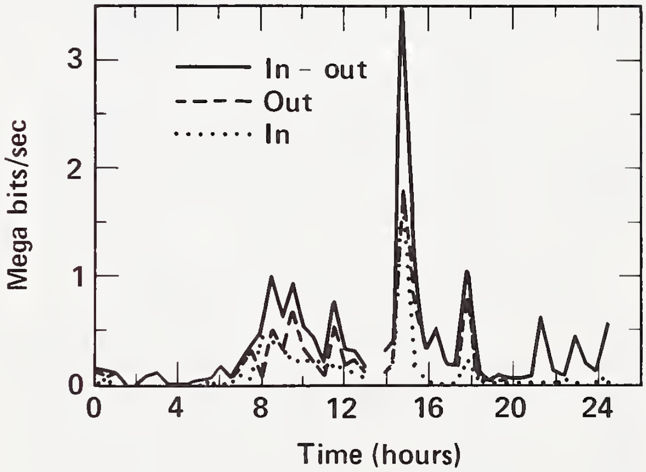


Figure 16. The total file transfer, in mega bits/second, into out of the CRAY-C machine to/from MASS and ATL combined.

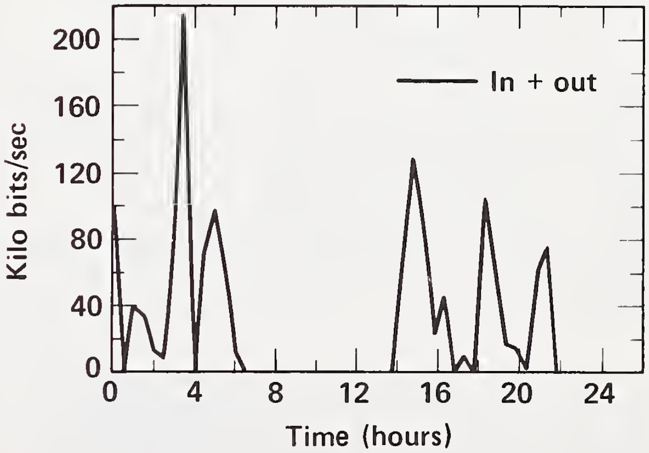


Figure 17. The total file transfer, in kilo bits/second, to the CHORS from the CRAY-C machine.

Table 2 contains a summary of this data and similar data obtained for the other Cray computers. In every instance, peak values are used. This table indicates the degree to which we understand each path. What is not depicted in the figures and tables so far is the distribution in time and size of the messages on these paths. It is not depicted because we do not know it. We do know the maximum, minimum and type of the messages on most of these paths. We do know, figures 13 to 17, qualitatively how the load/utilization varies during the day on most of these paths.

4.2 Medium monitor

If we had the Hyperchannel monitor device, the HMD described earlier, how would we use it and what sense could we make of the data collected by it?

The measurement process is simply described. The HMD is enabled, allowed to collect as much data as required to achieve statistically meaningful results, and then the collected data is analyzed.

The tape is written with the selected portions of 390 frames per record. It represents therefore a time history of the state transitions of every active adapter in the network. To be more specific, the header of each frame contains the destination adapter address, the source adapter address, frame function and length of data associated with the frame. These four pieces of information together with the time-stamp allow us to accurately follow the state transitions of every active adapter in the network. Using the transmission protocols given in figures 18a and 18b, it is possible to determine the beginning, length and end of each message transmitted in the network as well as its path (source/destination address pair). For a large sample of messages, we could expect to determine the message throughput and delay associated with each path through the network at any moment of the day. However, this point needs elaboration.

For the sake of the discussion that follows, let us define the Hyperchannel monitor (HM) as consisting of the HMD together with an output tape and the tape analysis program. Barring collisions, for any given frame from some source adapter, there is usually a corresponding response frame. A response frame can either be favorable (an ack) or unfavorable (a reject). In the case of a definite response, the HM will detect it and modify the state of the adapters involved. The lack of a response will be inferred by the HM by observing the behavior of the unsatisfied adapter, e.g. it will retry the unacknowledged frame. After some suitable number of retries, an adapter will abort the message transmission. Aborts can occur at any point in the transmission protocol, and the HM can detect (infer) them because of the transmitting adapter's deviation from it. A host may reattempt the message transmission upon receiving an abort indication from its attached adapter.

Eventhough the HM is unable to distinguish between new and reattempted messages, it does know not to count the bits of an aborted, partially transmitted message. Consequently, while the HM can determine the network utilization, it can not determine the message delay (host to host transmission delay). It can determine the message transmission delay associated with successful transmissions only. We could think of this as the "transmission delay", conditioned by the number of message aborts. It may be possible depending on the protocol involved to know the length of an aborted message.

Once the path behavior is sufficiently quantified, it would be possible through various statistical methods to determine such things as if and when there is any mutual interactions between paths.

So far, the network monitoring software resides in the Cray OSs only, and there are no plans to add a similar facility to the other hosts in the Craynet. In addition to this, since the monitoring software is resident at all times in the Cray OSs, efficiency and space considerations severely limit its capabilities and frequency of execution. Consequently, measurements are reported for half hour collection periods. That is to say, finer granularity is not possible without impacting host performance in unacceptable ways. It's fair to say that half hour wide collection periods do not provide a characterization of the instantaneous behavior of a network. Till now, our use

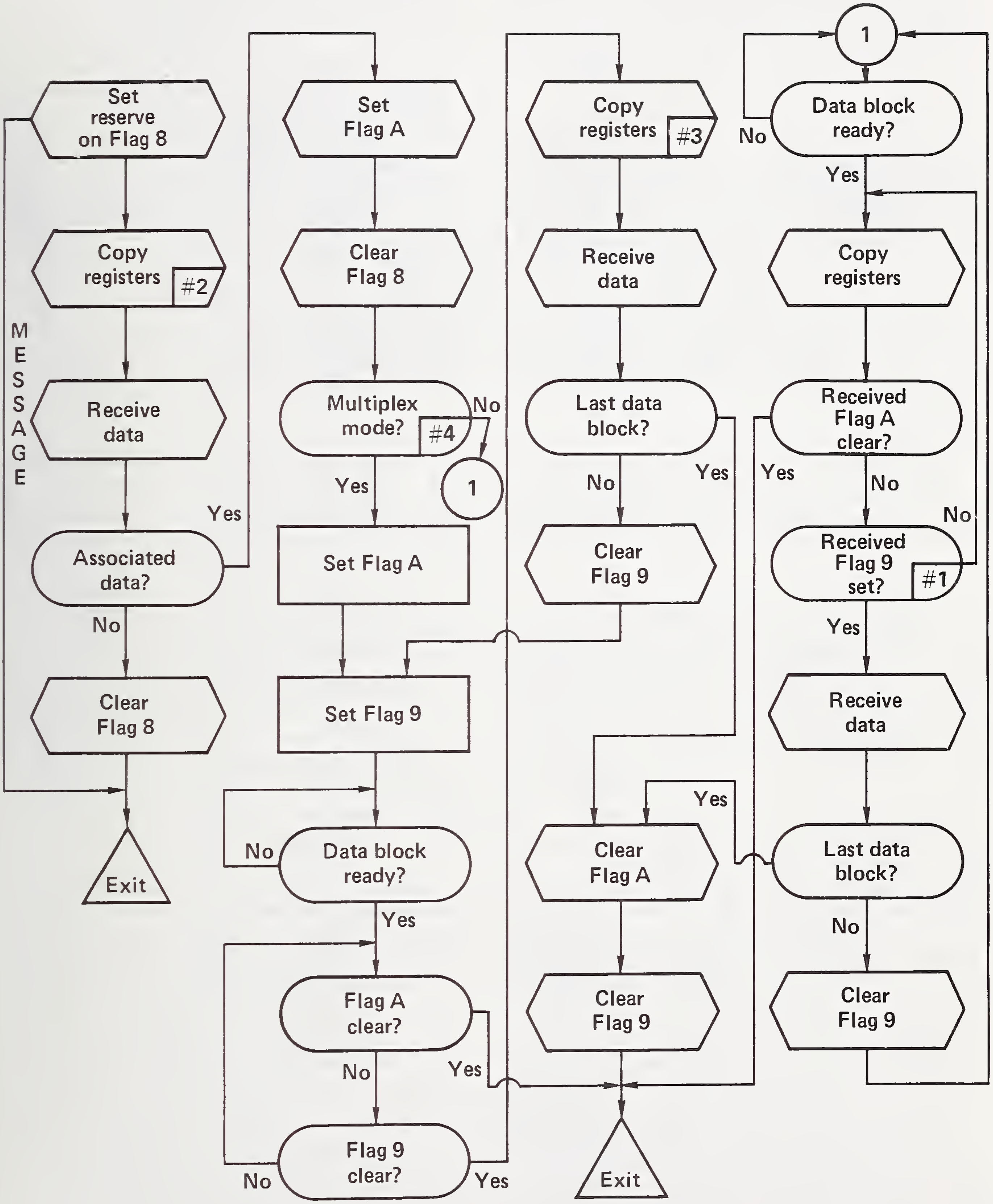


Figure 18a. The transmitter protocol.

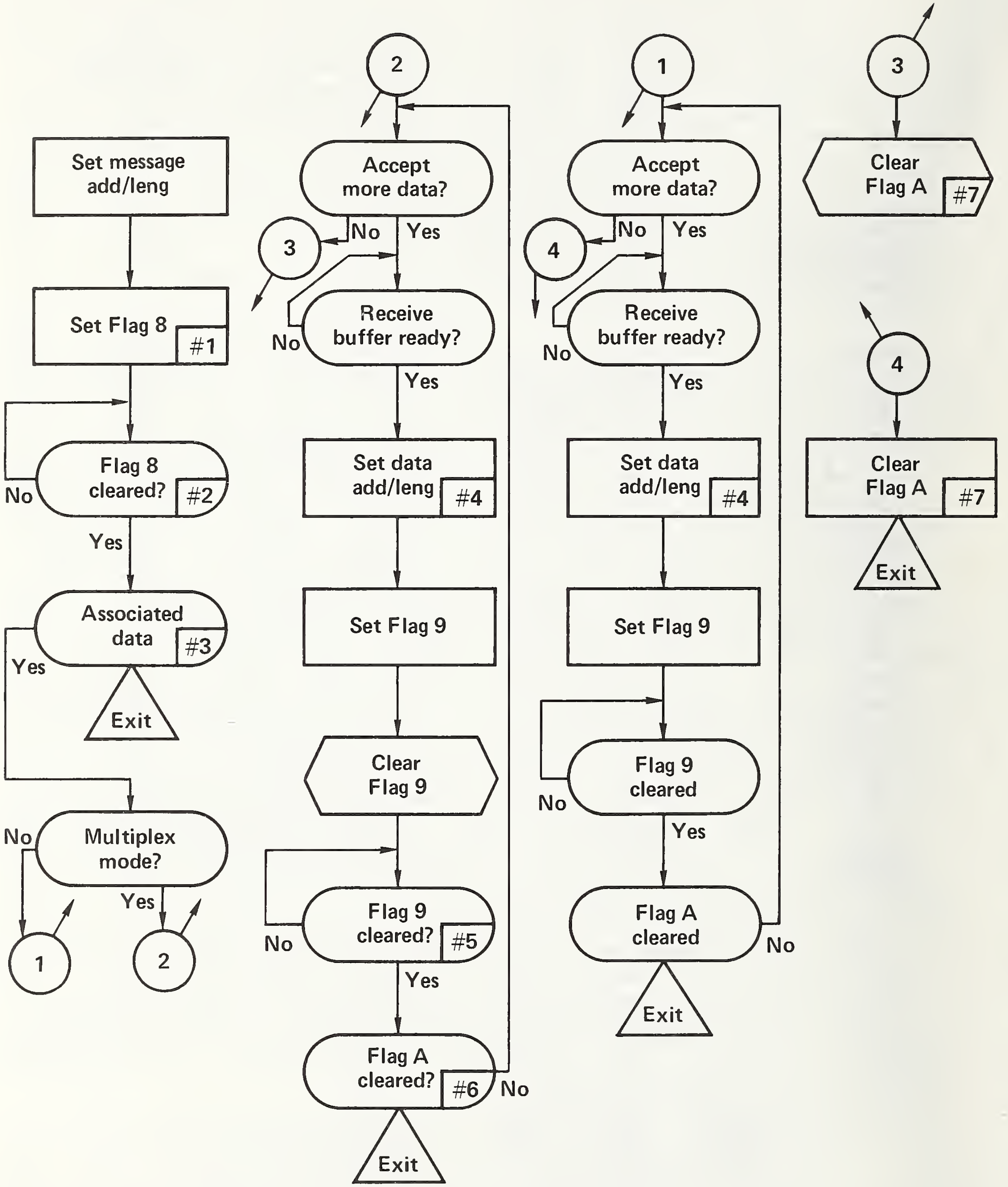


Figure 18b. The receiver protocol.

of monitoring software is at best incomplete (not resident in all network hosts) and too coarse to allow proper characterization of network traffic with the detail required for the NBS study.

With respect to the medium monitoring approach, as of this writing, our outside vendor has not yet completed work on the Hyperchannel monitor. This $175,000 piece of equipment, originally to be built in six months as part of a research contract with and funded by the NBS, is 13 months past scheduled completion. And because it is 13 months behind schedule, the success of this research project has been seriously jeopardized. Not only that, the absence of this monitoring equipment has eroded the case for the design-model-measure methodology I have been trying to build within my own group.

In any event, whereas we suffered basically from too little data using the software monitor approach, we will probably suffer from too much data using the medium monitor approach. In ten minutes of monitoring the Hyperchannel under a 5% load, enough data will be collected to fill a standard size magnetic tape. It will probably take more than an hour to analyze this much data using something like a Digital Equipment Corporation VAX/780. This disparity between sample width and analysis time will prove to be a troublesome but not insurmountable obstacle in the successful and timely completion of the NBS research project.

5. Model Validation

The *model*, which simulates Network Systems Corporation's (NSC) Hyperchannel, is the same one used extensively in the past to produce much useful knowledge pertaining to certain inadequacies and interactions of the lower level protocols, [4,5] inadequacies of the adapter management and buffering strategies,[6] and probed the nature of the interactions of network traffic, configuration, and these protocols.[7,8] We have validated this model to a certain extent[9] and so have great confidence in it. However, given the nature of the current investigation and the importance of the role the model will play, we feel that it is necessary to formally and extensively demonstrate the model's validity. Once we have determined that it accurately portrays network performance at low to medium loads, we will use the model to study performance and traffic characteristics at medium to high loads, a kind of computational extrapolation.

We resort to this extrapolation technique, because the Craynet as currently configured is not able to generate the level of traffic required for the complete characterization of network traffic and performance called for in the NBS study. Consequently, our approach will be to measure and characterize the traffic that exists in the current, operational Craynet in numerous data transfer situations, and then use the validated model to study its performance at higher loads.

The rest of this section will detail the process of model validation, briefly reviewing previous studies and papers exploring the possibility of using their results in this process. The validation process will make extensive use of the Hyperchannel Monitor Device (HMD).

The first step will be to determine that the sequence and timing of the frames of the lowest level protocol, figures 18a and 18b, exchanged in a simple two adapter configuration are exactly duplicated in sequence, and statistically and acceptably approximated in timing. This step can be performed on either the Cray network or a test network.

As a second step, the model will attempt to predict network performance in the three node, two data path configuration of figure 19. In the past, we have studied this configuration, known as the *message switch*, extensively because of its sensitivity to how adapter buffers are managed and to certain aspects of the lowest level protocols.[6,7,8,9] Specifically, the high data transfer path will operate at its maximum rate, while the slow path will attempt to transmit at various, slower rates. Simulation results[7] have shown that the fast and slow paths interfere with each other under these circumstances. We will determine the message throughput and delay of the two paths experimentally, using the HMD and/or processes residing in the attached hosts. Again simulated performance must

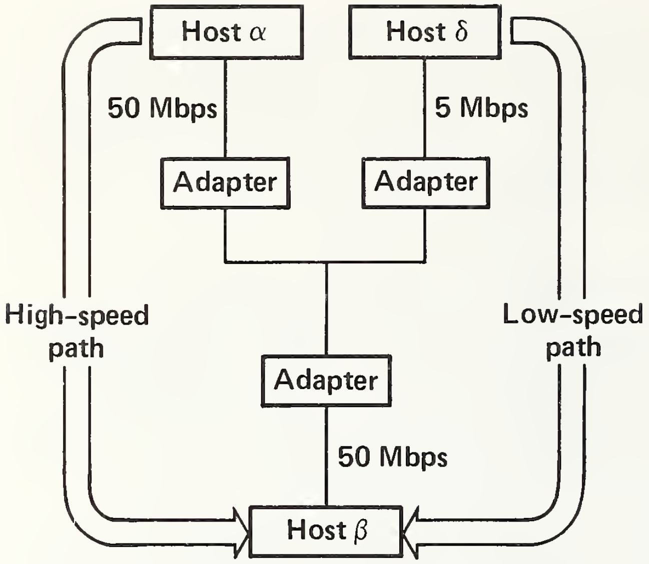


Figure 19. Uni-directional message switch.

approximate measured performance in an acceptably statistical way. We will probably perform this step in a test network owing to the impossibility of dedicating the Craynet to experiments even of brief duration. A test network will probably consist of three or four nodes and one bus, where the nodes will consist of a 7600 PPU/A120, a PDP11/A400, a SEL/A470, or a VAX/A400. However, it may be possible to conduct the message switch and other experiments on the Craynet during periods of minimal activity, collect the data with the HMD and subsequently identify and verify those periods of time within the sample in which the strict constraints of the original experiment were met.

The last step in the model validation process will be to measure and model the sequence and timing of exchanged frames in a four node, two path configuration. Again, the goal is to verify that the model exactly duplicates the sequence of exchanged frames and suitably approximates their timing. Experiments involving more nodes and buses are probably not possible since we can not intefere with the normal functioning of the Craynet.

In a previous paper,[7] I reported on the effect of segregating messages according to size, to wit, long and short messages were each transmitted over a bus dedicated to their respective sizes. The hypothesis was that somehow these two sizes (and types) of messages interfered with each other. The results reported in [7] do not support such a hypothesis, in fact, they indicate the contrary, i.e. the failure to segregate messages in this manner has no effect on network performance. It will not be possible to validate these results[7] using the HMD owing to the impossibility either of reproducing the configuration in [6], a five node two bus network, as a test network, or of modifying the host systems in the Craynet to behave in this manner.

Yeh and Donnelley[5] modelled the NSC Hyperchannel, (protocols and hardware), as it existed in 1978. Their simulations detected deadlocks and node priority reversals. Because NSC has long since remedied these shortcomings through modifications of adapter architecture and microcode, I see no possible way of using the HMD either in a test network or in the Craynet itself to validate their results.

Yeh and Donnelley[4] studied their own solution to these deadlock/reversal problems, viz. the recycling timer, and compared its effects to those produced by NSC's modifications, viz. wait flags and a binary exponential retry mechanism. No Hyperchannel experiment could validate the results obtained using their solution since this would require extensive modifications of Hyperchannel architecture and microcode.

Model validation and performance monitoring share many of the same problems. In order to validate a model, the thing modelled must be subjected to experiments in which it is controlled in some precise way, a way that can be duplicated by the model. In the case of networks, this usually entails modifying the operating system of the hosts involved in the validation experiment so that they use the network in some computationally meaningful way, e.g. submitting messages to the network that have nice distributions in size and interarrival time. Not only must the experimenter control the presented load precisely, he must also have some way of measuring the network's response to the load with equal precision. Again, one can either use a medium monitor and/or system resident software to do this. If one decides to use system resident software, he is presented with the additional problem of how to synchronize the various hosts in an experiment so that they all use the same time interval. If one relies instead upon some kind of a hardware monitor, he may be prevented from knowing certain aspects of the network's performance, e,g, message delay, that ultimately translates into an only partial validation of the model.

Even though we do not yet have a monitor, we have been able to validate our model using system resident software.[9] The problems here were finding host computers that could be diverted from their normal function long enough to conduct an experiment, and finding a willing systems programmer for each host involved. I was able to do both, and in our one and only set of validation experiments, the message switch, figure 19, alluded to earlier, we strung together two CDC-7600 PPUs and a Systems Engineering Laboratory (SEL) 32/75 and their appropriate adapters. The operating systems in both the PPU and SEL computers is written in assembly language. Fortunately, the PPU operating system had already been modified for use as part of the Hyperchannel acceptance test procedure. All that remained was to modify the operating system of the SEL host. This was done, experiments were run subsequently and the results are reported in [9]. Given the great paucity of statistical routines in both operating systems (no random number generators, mean or variance calculations), and that they were written in assembly language, and that both the system programmer and network technician were volunteering their 'spare' time to this effort, we had to settle for less than scientifically definitive, but nonetheless gratifying, results.

A truly and scientifically definitive validation of our model occurred almost incidentally when Franta and Heath[10] at the University of Minnesota validated their own analytical model of the Hyperchannel. They were able to realize the ideal in model validation experiments, for they had at their disposal a 6 node Hyperchannel network over which they had total control—they provided their own host system software; they were able to modify the adapter microcode to suit their needs; they had the use of a passive, medium monitor; and they had absolute control of and knowledge of the load presented to their test network. In addition to validating their own model, they were intent on studying the performance of the Hyperchannel, identifying its critical parameters and metrics.

It was a simple task for us to simulate their experiments and compare our results with theirs.[9,10]

And at what cost all of this? The software alluded to above required about two and a half man years. All Hyperchannel hardware, microcode modifications and engineering support were supplied by Network Systems Corporation free of charge. And as for actual funds used then, I do not know nor am allowed to know or report the source or the amount of the research grant supporting this project. However, I have been told that within university environments and as grants go, this one was small.

6. Extrapolate

As I have explained earlier, the Craynet has never been able to generate loads greater than 5% of Hyperchannel capacity with normal traffic on a sustained basis. As a consequence, we do not have and will probably never have any real experience with Hyperchannel performance at medium to high loads (> 50% of capacity, what I have termed hypothetical or extrapolated loads). However, we have already used the model to study its behavior at these hypothetical or extrapolated loads.[4,5,6,7,8,9] In fact, most of the shortcomings and inadequacies of the Hyperchannel alluded to throughout this paper occur only during such loads. Much of this early work represents what I have termed extrapolation—the exploration of realms of performance unreachable by means other than computer simulation. This is the more or less traditional use to which computer simulation is put, and presumably what is learned thereby helps one subsequently to actualize designs capable of such performance.

Extrapolation can be used in another sense here, not just in regard to studying performance at hypothetical loads of existing and/or experimental designs. The other sense of the word has to do with modelling or simulation itself. The layered nature of network functionality, both hardware and protocols, lends itself naturally to an incremental model contruction technique.[11] That is, once we have confidence in the simulation and modelling of the lower layers, we are able to add onto it models and simulations of higher layer functionality. Easier said than done. One may be justified in adding new layers to the model in that he has great confidence in its validity, but most computers and operating systems could care less how justified you feel or how much confidence you have in your own programs. Once your program no longer fits in memory, it no longer runs. Once your program grows to monstrous complexity, such large amounts of real time are required to produce results that it begins to lose significance as a part of the design-model-measure paradigm.

The growth of complexity and use of computer resources in modelling increasing layers of protocols stems directly from the mutual interaction of these layers themselves and the impact of this interaction on overall performance. For example, in the Hyperchannel each host to host message requires the sending host to undergo a three step interaction with its Hyperchannel adapter before the data (the original message) can cross the interface between them. Once that step is accomplished, the sending host's adapter and receiving host's adapter go through a minimum of 8 steps to actually move the data between them. The receiving adapter must then go through some protocol to move the data it's just received (the original message) up into the receiving host. A considerable amount of amplification can result as a message passes downward through layers and layers of protocols both in the number of bits (lower layers encapsulate higher layer's data in passing it downward to the next layer) and in the number of interactions (hand shakes, buffer negotiations, exchanges of remembered state, acks etc.).

How does the modeller/simulationist proceed in this matter given this fact of life? One can proceed by means of what I've termed extrapolation in which that part of a model dealing with the lower layers is replaced by something simpler that imitates (simulates?) the behavior of the lower layers. To be more specific, our approach will use data derived from Hyperchannel monitor studies to characterize Hyperchannel behavior, i.e. to determine the message delay distribution as a function of path, message size and ambient load within the network. We will also use the existing model to tell us what kinds of message delay distributions occur at extrapolated loads. Then we will build a new model that simulates the functionality of host to host message traffic and the appropriate higher layers of protocols, a model in which we replace the lowest layers of protocols and hardware (the Hyperchannel itself) with these delay distributions.

7. Conclusion

I have described a cyclic, design-measure-model methodology currently being used to study a large, Hyperchannel based LAN at the Lawrence Livermore National Laboratory. This NBS sponsored project expects to characterize this network's traffic and corresponding performance. However, many questions remain concerning how best to proceed in this matter.

References

[1] Clark D. D., Progran K. T., and Reed D. P., An Introduction to Local Area Networks, Proceedings of the IEEE, Special Issue on Packet Communications Networks, November, 1978, pp. 1497-1517.

[2] Sunshine C. A., Factors in Interprocess Communication Protocol Efficiency for Computer Networks, AFIPS Conf. Proc., NCC, vol 45, June 1976

[3] Pouzin L., and Zimmermann H., A Tutorial on Protocols, Proceedings of the IEEE, Special Issue on Packet Communications Networks, November, 1978, pp. 1346-1371.

[4] Donnelley, J. E. and Yeh J. W., Simulation Studies of Round Robin Contention in a Prioritized Broadcast Network, University of California, Lawrence Livermore Laboratory UCRL-81715, 1978 also Proceedings of the Third

Minnesota Conference on Local Networks, 1978.

[5] Donnelley, J. E. and Yeh J. W., Interaction Between Protocol Levels In a Prioritized CSMA Broadcast Network, University of California, Lawrence Livermore Laboratory UCRL-81151, 1978. Also, Proceedings of the Third Berkeley Workshop on Distributed Data Base Management and Computer Networks, 1978.

[6] Nessett D. M., Protocols for Buffer-Space Allocation in CSMA Broadcast Networks with Intelligent Interfaces, University of California, Lawrence Livermore Laboratory UCRL-81687, 1978. Also, Proceedings of the Third Minnesota Conference on Local Networks, 1978.

[7] Watson W. B., Simulation Study of the Traffic Dependent Performance of a Prioritizied, CSMA Broadcast Network, Computer Networks, 3, 1979, pp.427-434.

[8] Watson W. B., Simulation Study of the Configuration Dependent Performance of a Prioritizied, CSMA Broadcast Network, Computer, 14, No. 2, February, 1981, pp. 51-58.

[9] Watson W. B., Validation of a Discrete Event Computer Model of Network Systems Corporation's Hyperchannel, Proceedings of the 7th Conference on Local Computer Networks, Minneapolis, Minnesota, 1978, pp. 44-49.

[10] Franta W. R., Heath J. R., Performance of Hyperchannel Networks: Parameters, Measurements, Models and Analysis, University of Minnesota Computer Science Department Technical Report 82-3, January, 1982.

[11] Yeh J. W., Simulation of Local Computer Networks—a Case Study, Computer Networks, 3, 1979, pp.401-417.

QUEUE LENGTH CHARACTERISTICS AT VERY FAST, CONSTANT SERVICE TIME MERGER NODES

Chaim Ziegler
Brooklyn College
Dept. of Computer & Information Science
Bedford Ave. & Ave. H
Brooklyn, N.Y. 11210

Abstract:

This paper concerns itself with the determination of the steady-state queue length distributions at very fast merger nodes that are present within queueing networks. We study a network with a tree topology in which a given server at a node of the network provides each of its customers with an equal, constant amount of service time. A very fast merger node is defined as a node at which the service rate is greater than or equal to the sum of the service rates of the channels feeding into the merger node. Inputs following general, independent probability distributions are considered. A condition for absolute stability at the merger node is derived. The exact queue length distribution is found for a subset of these fast merger nodes via a combinatorial analysis of possible arrival patterns of customers into the merger node.

I. Introduction:

This paper concerns the study of the state probabilities of the queueing network shown in Figure 1. In Figure 1, each node shown represents an infinite storage, FCFS queueing facility equipped with a single server that has a constant service time of μ_i (customer/sec) (or service time of s_i sec/customer) i=1,2,...,n.

We assume that the network is being fed by "n" mutually independent streams of customers governed by some general probability distribution with mean arrival rate of λ_i (packets/sec.) for the class i stream, i=1,2,...,n, $\lambda_i \geq 0$. Each customer is considered to consist of a single, fixed length packet of data which is to be transmitted through the network. Class i packets initially enter the network at node N_i where they are then processed and transmitted to node N_{n+1} for additional processing. At node N_{n+1}, all n packet streams are merged and processed by a single common server. Hence, node N_{n+1} is called a merger node.

This paper is directed towards obtaining various queue length characteristics at the merger node, N_{n+1}. The discussion is limited to the case where the service rate at N_{n+1}, μ_{n+1}, is greater than or equal to the sum of the service rates of the feeder channels (i.e., $\mu_{n+1} \geq \sum_{i=1}^{n} \mu_i$ or $\frac{1}{s_{n+1}} \leq \sum_{i=1}^{n} \frac{1}{s_i}$). In earlier work [1,2], it has been shown that condition on μ_{n+1} is a sufficient condition for absolute stability at the merger node when n = 2. Here, we extend this result to general values of "n". In addition, in those papers, for the subcase $s_{n+1} \leq 1/n \min (s_1, s_2, \ldots, s_n)$ (or $\mu_n \geq n \max (\mu_1, \mu_2, \ldots, \mu_n))$, n=2, the steady-state waiting time statistics were derived. Here, we turn to the queue length probabilities and once again derive results for general values of "n".

II. Stability Condition for Very Fast Merger Nodes

Our study begins with a derivation of an interesting stability property for the case $\mu_{n+1} \geq \sum_{i=1}^{n} \mu_i$. In earlier work [1,2], it was shown that for the case n=2, the number of packets waiting for service on queue could never exceed one. This will now be generalized for arbitrary values of "n".

This work was partially supported by the National Science Foundation under grant ECS-8105963 and by PSC-CUNY Research Award 13665.

Theorem 1:

For the case $\mu_{n+1} \geq \sum_{i=1}^{n} \mu_i$, at the merger node

$$W \leq (n-1)\ s_3 \tag{1a}$$

$$N_q \leq n-1 \tag{1b}$$

where the quantities W and N_s are defined as:

$W \triangleq$ The waiting time (queueing time) in an arbitrary packet at N_{n+1}

$N_q \triangleq$ the number of packets on queue at N_{n+1}

Proof:

The proof will be via a worst case analysis which occurs when μ_{n+1} is set to its minimum value; that is, $\mu_{n+1} = \sum_{i=1}^{n} \mu_i$. This value of μ_{n+1} yields the greatest probability for the queue length to grow.

With $\mu_{n+1} = \sum_{i=1}^{n} \mu_i$, without loss of generality one can sort the "n" parallel feeding channels into ascending order of magnitude and obtain

$$s_i = k_i\ s_{n+1}\ ;\quad i=1,2,\ldots,n \tag{2a}$$

where

$$\left[1-\sum_{j=1}^{i-1}(1/k_j)\right]^{-1} \leq k_i \leq (n+1-i)\left[1-\sum_{j=1}^{i-1}(1/k_j)\right]^{-1} \tag{2b}$$

and

$$\sum_{i=1}^{n}(1/k_i) = 1 \tag{2c}$$

A worst case arrival pattern into N_{n+1} is now constructed by assuming that class i packets are arriving into N_{n+1} at regular intervals of s_i seconds. Thus, with

$t_{ij} \triangleq$ the arrival time into N_{n+1} of the j^{th} class i packets, i=1,2,...,n,

it follows that

$$t_{ij} = \left[x_i+(j-1)\right]k_i s_{n+1}\quad ;\quad i=1,2,\ldots,n\ ;\quad j=1,2,\ldots,0\leq x_i<1 \tag{3}$$

where x_i is a fractional offset from time zero for the first class i arrival.

An induction procedure will now be used to prove the theorem. This will be done by introducing the notation,

$W_{ij} \triangleq$ the waiting time at N_{n+1} of the j^{th} class i packet, and showing that $W_{ij} \leq s_{n+1}$ for all i and j.

To begin, Theorem 1 is trivially true for the case n = 1. (In fact, in [1,2], it has been proven true for n = 2.) It is now assumed that Theorem 1 is true for "n-1" channels feeding the merger node. It must now be shown that this implies validity of theorem 1 for "n" channels feeding N_{n+1} (as shown in Fig.1).

With "n" feeding channels, it can be stated that

$$W_{i1} \leq (n-1)\ s_{n+1};\quad i=1,2,\ldots,n \tag{4}$$

This is true because until the arrival of the first class i packet, node N_{n+1} sees an arrival pattern from at most n-1 classes of packets (excluding the class i stream) with s_{n+1} less than the parallel combination of the service times of the feeding channels. Thus, (4) is true by the induction assumption.

To complete the proof, we need only show $W_{ij} \leq (n-1)s_n$ for all j. We need concern ourselves only with the case where the busy period has been continuous from t_{i1} through t_{ij}. Otherwise, a renewal point of Process m would have occurred, where Process m is defined as the process whereby a class m packet initiates the current busy period m=1,2,...,n.

To derive an expression for W_{ij}, it is observed that in the interval (t_{i1}, t_{ij}), there will be j-2 additional class i arrivals and $\left\lfloor\frac{t_{ij}-t_{i1}}{k_m s_{n+1}}\right\rfloor$ class m arrivals. Consequently,

$$W_{ij} = \left[(t_{ij}+W_{i1}+s_{n+1})+(j-2)s_{n+1} + \sum_{\substack{m=1\\ m\neq i}}^{n}\left\lfloor\frac{t_{ij}-t_{i1}}{k_m s_{n+1}}\right\rfloor s_{n+1}\right] - t_{ij} \tag{5}$$

$$\leq \Bigg[(t_{i1}+W_{i1}+s_{n+1})+(j-2)s_{n+1} + \sum_{\substack{m=1\\ m\neq i}}^{n} (t_{ij}-t_{i1})/(k_m s_{n+1}) \Bigg] -t_{ij}$$

$$= W_{i1} + (j-1)s_{n+1} - (j-1)k_i s_{n+1} + \sum_{\substack{m=1\\ m\neq i}}^{n} \frac{(j-1)k_i s_{n+1}}{k_m}$$

$$= W_{i1} + (j-1)s_{n+1} - (j-1)k_i s_{n+1} + (j-1)k_i \left[1-\frac{1}{k_i} \right] s_{n+1}$$

$$= W_{i1} \leq (n-1)s_{n+1}$$

where we have use of both (2c) and (4).

This proves (1a). Equation (1b) follows directly from (1a). Q.E.D.

Theorem 1 has direct implications on buffer size requirements at a very fast merger node.

III. The Case $s_{n+1} \leq 1/n \min(s_1,s_2,\dots,s_n)$ (or $\mu_{n+1} \geq n \max(\mu_1,\mu_2,\dots,\mu_n)$)

For the case $s_{n+1} \leq 1/n \min(s_1,s_2,\dots,s_n)$, it is intuitive that the number of packets serviced within a busy period at N_{n+1} can also not exceed "n". (For example, if "n" packets arrived in a common interval of s_{n+1} seconds, then a second packet belonging to any class can not arrive into N_{n+1} for at least ns_{n+1} seconds from the arrival time of its first packet. By that time, all "n" original packets must already have been serviced at N_{n+1} and, hence, the busy period ended.) Furthermore, each packet serviced in a busy period must come from a different class.

The following theorem, concerning the queue length probabilities at node N_{n+1}, can now be stated

Theorem 2:

For the case $s_{n+1} \leq 1/n \min(s_1,s_2,\dots,s_n)$, the equilibrium queue length distribution at a random instant of time at node N_{n+1} of Figure 1 is given by

$$P_i = \sum_{j=0}^{n} \sum_{k=j}^{n} \frac{C_{ij} \binom{k}{j} (-n)^{k-j} V_k}{\binom{n}{j}} \tag{6}$$

where

$P_i \triangleq$ the equilibrium probability that node N_{n+1} contains "i" packets (both on queue and in service), $i=0,1,\dots,n$;

$C_{ij} \triangleq$ the total number of arrival patterns that will leave "i" packets in the system at a random observation epoch, given that there were "j" arrivals in the $(n)(s_{n+1})$ seconds directly preceding the observation epoch;

$$= \binom{n}{j} \sum_{L=0}^{j-i} \Bigg[\sum_{\substack{x_k=0\\ k=1,2,\dots,L}}^{y_k} \cdots \sum_{\substack{x_k=0\\ k=L+1,L+2,\dots,j-i-1}}^{z_k} \tag{6a}$$

$$\left[\prod_{k=1}^{j-i-1} \binom{j-\sum_{m=1}^{k-1} x_m}{x_k} \right] \Bigg] \binom{j-\sum_{m=1}^{j-i-1} x_m}{i+L-\sum_{m=1}^{L} x_m} (n-j+i)^{j-i-L-\sum_{m=L+1}^{j-i-1} x_m} \Bigg]$$

where

$$y_k = i-2+k-\sum_{m=1}^{k-1} x_m; \qquad k=1,2,\dots,L$$

$$z_k = k-L-\sum_{m=1}^{k-L-1} x_m; \qquad k=L+1,L+2,\dots,j-i-1$$

(Note: see Appendix A for the derivation of C_{ij})

$V_k \triangleq$ the sum of the products of ρ_i taken "k" at a time, $k=0,1,\dots,n$; that is,

$V_0 = 1$

$V_1 = \sum_{i=1}^{n} \rho_i$

$$V_2 = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\rho_i\rho_j$$

$$\vdots$$

$$V_{n-1} = \sum_{i=1}^{2}\sum_{j=i+1}^{3}\cdots\sum_{k=n-1}^{n}\rho_i\rho_j\cdots\rho_k$$

$$V_n = \prod_{i=1}^{n}\rho_i$$

where $\rho_i = \lambda_i s_{n+1}$, $i=1,2,\ldots,n$.

Proof:

To begin the proof, Figure 2 illustrates the random instant of time at which an observation is to be made of the state of the system at node N_{n+1}.

It has already been shown that the total number of packets resident at N_{n+1} can never exceed "n". This fact, implies that any packet present in the system at our observation epoch must have arrived within the immediately preceding ns_{n+1} seconds. Consequently, to illustrate this in Figure 2, a time line has been drawn of length ns_{n+1} seconds extending back from our random observation epoch (random time t). In addition, the time line has been partitioned into "n" equal intervals (or slots), each of length s_{n+1} seconds.

A pause is taken to note that since $s_{n+1} \leq 1/n \min(s_1,s_2,\ldots,s_n)$, the maximum length of a busy period must be ns_{n+1} seconds. Hence, a renewal point, which for our purposes is a conclusion of a busy period, must occur within the ns_{n+1} seconds depicted in Figure 2.

Continuing with the proof, it is noted that since $s_{n+1} \leq 1/n \min(s_1,s_2,\ldots,s_n)$, over the interval $(t-ns_{n+1},t)$, at most one packet from each of the "n" different classes of packets arrived into N_{n+1}. In addition, if a class "i" packet arrives within this interval, it arrives with equal probability within any of the "n" slots shown in Figure 2. This is true simply because our observation epoch is completely random. Consequently, the total number of ways for "j" packets to arrive at N_{n+1} within the "n" slots preceding the observation epoch, drawing from our population of "n" different classes of packets, is

$$\binom{n}{j}n^j \tag{7}$$

Now, with C_{ij} defined as above, it can be concluded that the probability of observing "i" packets in the system at the observation epoch conditioned on the event that "j" packets arrived in the last "n" slots is

$$\frac{C_{ij}}{\binom{n}{j}n^j} \tag{8}$$

Noting that the probability that a class "i" packet arrives within any particular slot is given by

$$\rho_i = \lambda_i s_{n+1}; \quad i=1,2,\ldots,n \tag{9}$$

and that the probability that a class "i" packet arrives over the interval $(t-ns_{n+1},t)$ is

$$n\rho_i = n\lambda_i s_{n+1}; \quad i=1,2,\ldots,n \tag{10}$$

it follows that the probability of having "j" arrivals over the "n" slots of the interval $(t-ns_{n+1},t)$ is given by

$$\begin{aligned}
&(n\rho_1)(n\rho_2)\ldots(n\rho_j)(1-n\rho_{j+1})(1-n\rho_{j+2})\ldots(1-n\rho_n)\\
&+ (n\rho_1)(n\rho_2)\ldots(n\rho_{j-1})(1-n\rho_j)(n\rho_{j+1})(1-n\rho_{j+2})\\
&\qquad\ldots(1-n\rho_n)\\
&\vdots\\
&+ (n\rho_1)(n\rho_2)\ldots(n\rho_{j-1})(1-n\rho_j)(1-n\rho_{j+1})\\
&\qquad\ldots(1-n\rho_{n-1})(n\rho_n)\\
&\vdots\\
&+ (1-n\rho_1)(1-n\rho_2)\ldots(1-n\rho_{n-j})(n\rho_{n-j+1})\ldots(n\rho_n)\\
&= \sum_{k=j}^{n}\binom{k}{j}(-1)^{k-j}(n)^k V_k
\end{aligned} \tag{11}$$

where V_k is as defined above. This result, (11), was derived by calculating all possible combinations of having "j" different arrivals from the "n" possible classes over the indicated interval.

Of consequence, using (3) and (6), it is found that the joint probability of observing "i" packets in the system at the observation epoch AND having "j" arrivals within the last "n" slots is given by

$$\left[\frac{C_{ij}}{\binom{n}{j}n^{j}}\right]\left[\sum_{k=j}^{n}\binom{k}{j}(-1)^{k-j}(n)^{k}V_{k}\right] \qquad (12)$$

$$= C_{ij}\sum_{k=j}^{n}\frac{\binom{k}{j}(-n)^{k-j}V_{k}}{\binom{n}{j}}$$

Finally, to conclude the proof, the unconditional probability of observing "i" packets in the system at the observation epoch is obtained. This is done by summing (12) over all possible "j". The result is

$$P_{i} = \sum_{j=0}^{n}C_{ij}\sum_{k=j}^{n}\frac{\binom{k}{j}(-n)^{k-j}V_{k}}{\binom{n}{j}} \qquad (13)$$

which proves the theorem. Q.E.D.

With a bit of tedious algebra, one can now proceed to determine the steady-state mean characteristics at N_{n+1}. Defining

$\overline{N} \triangleq$ the steady-state mean number of arbitrary class packets in the system at node N_{n+1},

$\overline{N}_{q} \triangleq$ the steady-state mean number of arbitrary class packets on queue at node N_{n+1},

one finds

$$\overline{N} = \sum_{i=0}^{n} iP_{i} = V_{1}+\sum_{k=2}^{n}\frac{k!}{2}V_{k} \qquad (14a)$$

and

$$\overline{N}_{q} = \sum_{i=0}^{n-1} iP_{i+1} = \sum_{k=2}^{n}\frac{k!}{2}V_{k} \qquad (14b)$$

which are pleasing results.

Little's Formula [3] can now be used on (14a) and (14b) to find the steady-state mean response time and mean queueing time respectively. (This extends the results of our earlier papers [1,2) to general n.)

To complete this section, the queue length distributions for a specific class packet at the merger node is derived. To do this, we define

$P_{i}^{(j)} \triangleq$ the steady-state probability of having "i" class "j" packets in the system at the merger node, i=0,1; j=1,2,...,n

$P_{q(i)}^{(j)} \triangleq$ the steady-state prbability of having "i" class "j" packets on queue at the merger node, i=1,2; j=1,2,...,n.

$P_{i}^{(j)}$ and $P_{q(i)}^{(j)}$ can be derived using a similar approach as that used in the proof of Theorem 1. However, a much simpler derivation is possible if one looks at (14a) and (14b) and realizes that

$$\overline{N} = \sum_{j=1}^{n} P_{1}^{(j)} \qquad (15a)$$

and

$$\overline{N}_{q} = \sum_{j=1}^{n} P_{q(1)}^{(j)} \qquad (15b)$$

Then, using the fact that not more than a single packet from each class can be processed over $(t-ns_{n+1},t)$, simple symmetry arguments yield

$$P_{1}^{(j)} = V_{1}' + \sum_{k=2}^{n}\frac{(k-1)!}{2}V_{k}', \qquad j=1,2,\dots,n \qquad (16a)$$

$$P_{0}^{(j)} = 1-P_{1}^{(j)}, \qquad j=1,2,\dots,n \qquad (16b)$$

and

$$P_{q(1)}^{(j)} = \sum_{k=2}^{n}\frac{(k-1)!}{2}V_{k}', \qquad j=1,2,\dots,n \qquad (17a)$$

$$P_{q(0)}^{(j)} = 1-P_{q(1)}^{(j)}, \qquad j=1,2,\dots,n \qquad (17b)$$

where V_{k}' is found by taking only those terms from V_{k} in which ρ_{j} appears and is defined as the product of ρ_{j} AND the sum of the products of the ρ_{i} taken "k" at a time, i=1,2,...,n; $i \neq j$; k=0,1,...,n-1.

IV. Conclusion:

In this paper, we have successfully derived the steady-state queue length distributions at a merger node for a certain class of network conditions. We calculated these results by an analysis of the arrival patterns of the input streams into the merger node. We believe this technique can be utilized in the study of a variety of network structures.

References:

[1] C. Ziegler and D.L. Schilling, "Waiting Times at Fast Merger Nodes," Conference Record of ICC '80, pp. 23.2-1 - 23.2-6, June 1980.

[2] C. Ziegler and D.L. Schilling, "Waiting Times at Very Fast, Constant Service Time Merger Nodes," accepted for publication in IEEE Transactions on Communications.

[3] J.D.C. Little, "A Proof of the Queueing Formula $L= \lambda W$," Oper. Res., Vol. 9 pp. 383-7, 1961.

Appendix A:

In this appendix, we derive the equation for C_{ij}. This will be done by first solving for some simple cases and then generalizing the results.

We begin by noting that

$$C_{ij} = 0, \quad \text{for} i>j \qquad \text{(A-1)}$$

Hence, we must only consider $0 \le i \le j \le n$.

For i=j, all "j" arrivals must occur within slot 1. Thus, remembering that there are "n" different classes of packets, we have

$$C_{j,j} = \binom{n}{j} \qquad \text{(A-2)}$$

For i=j-1, there are two possibilities; namely, having "j-1" arrivals in slot 1 and one arrival in any other slot OR having "i" arrivals in slot 1 and "j-i" arrivals in slot 2, i=0,1,...,j-2. Consequently,

$$C_{j-1,j} = \binom{n}{j-1} \binom{n-(j-1)}{1}(n-1) + \sum_{i=0}^{j-2} \binom{n}{i} \binom{n-i}{j-i}$$

$$= \binom{n}{j} \binom{j}{1}(n-1) + \binom{n}{j} \sum_{i=0}^{j-2} \binom{j}{i} \qquad \text{(A-3)}$$

Moving to the case i=j-2, we find that there are now three possibilities:

1) "j-2" arrivals occur in slot 1 AND "i" arrivals occur in slot 2 (i=0,1) AND "2-i" arrivals occur over the remaining "n-2" slots,

2) "i" arrivals occur in slot 1 (i=0,1,...,j-3) AND "j-i-1" arrivals occur in slot 2 AND one arrival occurs in any one of the remaining "n-2" slots,

3) "i" arrivals occur in slot 1 (i=0,1,...,j-3) AND "k" arrivals occur in slot 2 (k=0,1,...,j-i-2) AND "j-i-k" arrivals occur in slot 3.

This yields

$$C_{j-2,j} = \binom{n}{j-2} \sum_{i=0}^{1} \binom{n-(j-2)}{i} \binom{n-(j-2)-i}{2-i} (n-2)^{2-i}$$

$$+ \sum_{i=0}^{j-3} \binom{n}{i} \binom{n-i}{j-i-1} \binom{n-(j-1)}{1} (n-2)$$

$$+ \sum_{i=0}^{j-3} \binom{n}{i} \sum_{k=0}^{j-i-2} \binom{n-i}{k} \binom{n-i-k}{j-i-k}$$

$$= \binom{n}{j} \sum_{i=0}^{1} \binom{j}{i} \binom{j-i}{2-i} (n-2)^{2-i}$$

$$+ \binom{n}{j} \sum_{i=0}^{j-3} \binom{j}{i} \binom{j-i}{1} (n-2)$$

$$+ \binom{n}{j} \sum_{i=0}^{j-3} \sum_{k=0}^{j-i-2} \binom{j}{i} \binom{j-i}{k} \qquad \text{(A-4)}$$

In a similar manner, one can show that for i=j-3 there are four possibilities to consider which yield

$$C_{j-3,j} = \binom{n}{j-3} \sum_{i=0}^{1} \binom{n-(j-3)}{i} \sum_{k=0}^{2-i} \binom{n-(j-3)-i}{k}$$

$$\binom{n-(j-3)-i-k}{3-i-k} (n-3)^{3-i-k}$$

$$+ \sum_{i=0}^{j-4} \binom{n}{i} \binom{n-i}{j-i-2} \sum_{k=0}^{1} \binom{n-(j-2)}{k} \binom{n-(j-2)-k}{2-k} (n-3)^{2-k}$$

$$+ \sum_{i=0}^{j-4} \binom{n}{i} \sum_{k=0}^{j-3-i} \binom{n-i}{k} \binom{n-i-k}{j-1-i-k} \binom{n-(j-1)}{1} (n-3)$$

$$+ \sum_{i=0}^{j-4} \binom{n}{i} \sum_{k=0}^{j-3-i} \binom{n-i}{k} \sum_{L=0}^{j-2-i-k} \binom{n-i-k}{L} \binom{n-i-k-L}{j-i-k-L}$$

$$= \binom{n}{j} \sum_{i=0}^{1} \sum_{k=0}^{2-i} \binom{j}{i} \binom{j-i}{k} \binom{j-i-k}{3-i-k} (n-3)^{3-i-k}$$

$$+ \binom{n}{j} \sum_{i=0}^{j-4} \sum_{k=0}^{1} \binom{j}{i} \binom{j-i}{k} \binom{j-i-k}{2-k} (n-3)^{2-k}$$

$$+ \binom{n}{j} \sum_{i=0}^{j-4} \sum_{k=0}^{j-3-i} \binom{j}{i} \binom{j-i}{k} \binom{j-i}{1} (n-3)$$

$$+ \binom{n}{j} \sum_{i=0}^{j-4} \sum_{k=0}^{j-3-i} \sum_{L=0}^{j-2-i-k} \binom{n}{i} \binom{j-i}{k} \binom{j-i-k}{L} \qquad \text{(A-5)}$$

Proceeding recursively with this method, one finds that for the general term, i=j-q, we obtain

$$C_{j-q,j} = \qquad \text{(A-6)}$$

$$= \binom{n}{j} \sum_{i=0}^{1} \sum_{k=0}^{2-i} \sum_{L=0}^{3-i-k} \cdots \sum_{m=0}^{q-1-i-k-L-\ldots} \binom{j}{i} \binom{j-i}{k} \binom{j-i-k}{L} \cdots \binom{j-i-k-L\ldots}{m} \binom{j-i-k-L-\ldots-m}{q-i-k-L-\ldots-m} (n-q)^{q-i-k-L-\ldots-m}$$

$$+ \binom{n}{j} \sum_{i=0}^{j-q-1} \sum_{k=0}^{1} \sum_{L=0}^{2-k} \cdots \sum_{m=0}^{q-2-k-L-\ldots} \binom{j}{i} \binom{j-i}{k} \binom{j-i-k}{L} \cdots \binom{j-i-k-L\ldots}{m} \binom{j-i-k-L-\ldots-m}{q-1-k-L-\ldots-m} (n-q)^{q-1-k-L-\ldots-m}$$

$$+ \binom{n}{j} \sum_{i=0}^{j-q-1} \sum_{k=0}^{j-q-i} \sum_{L=0}^{1} \cdots \sum_{m=0}^{q-3-L-\ldots} \binom{j}{i} \binom{j-i}{k} \binom{j-i-k}{L} \cdots \binom{j-i-k-L\ldots}{m} \binom{j-i-k-L-\ldots-m}{q-2-L-\ldots-m} (n-q)^{q-2-L-\ldots-m}$$

$$\vdots$$

$$+ \binom{n}{j} \sum_{i=0}^{j-q-1} \sum_{k=0}^{j-q-i} \sum_{L=0}^{j-q-i-k+1} \cdots \sum_{m=0}^{1} \binom{j}{i} \binom{j-i}{k} \binom{j-i-k}{L} \cdots \binom{j-i-k-L\ldots}{m} \binom{j-i-k-L-\ldots-m}{2-m} (n-q)^{2-m}$$

$$+ \binom{n}{j} \sum_{i=0}^{j-q-1} \sum_{k=0}^{j-q-i} \sum_{L=0}^{j-q-i-k+1} \cdots \sum_{m=0}^{j-3-i-k-L\ldots} \binom{j}{i} \binom{j-i}{k} \binom{j-i-k}{L} \cdots \binom{j-i-k-L\ldots}{m} \binom{j-i-k-L-\ldots-m}{1}$$

$$+ \binom{n}{j} \sum_{i=0}^{j-q-1} \sum_{k=0}^{j-q-i} \sum_{L=0}^{j-q-i-k+1} \cdots \sum_{m=0}^{j-3-i-k-L\ldots} \sum_{p=0}^{j-2-i-k-L-\ldots-m} \binom{j}{i} \binom{j-i}{k} \binom{j-i-k}{L} \cdots \binom{j-i-k-L\ldots}{m} \binom{j-i-k-L-\ldots-m}{p}$$

which can be written as

$$C_{ij} = \tag{A-7}$$

$$\binom{n}{j}\sum_{L=0}^{j-i}\left[\underset{k=1,2,\ldots,L}{\sum_{x_k=0}^{y_k}}\cdots\underset{k=L+1,L+2,\ldots,j-i-1}{\sum_{x_k=0}^{z_k}}\left[\prod_{k=1}^{j-i-1}\binom{j-\sum_{m=1}^{k-1}x_m}{x_k}\right]\right.$$

$$\left.\cdot\binom{j-\sum_{m=1}^{j-i-1}x_m}{i+L-\sum_{m=1}^{L}x_m}(n-j+i)^{j-i-L-\sum_{m=L+1}^{j-i-1}x_m}\right]$$

where

$$y_k = i-2+k-\sum_{m=1}^{k-1}x_m; \qquad k=1,2,\ldots,L \tag{A-7a}$$

$$z_k = k-L-\sum_{m=1}^{k-L-1}x_m; \qquad k=L+1,L+2,\ldots,j-i-1 \tag{A-7b}$$

and

$$\underset{k=1,2,\ldots,L}{\sum_{x_k=0}^{y_k}}\cdots\underset{k=L+1,L+2,\ldots,j-i-1}{\sum_{x_k=0}^{z_k}} = \tag{A-7c}$$

$$\sum_{x_1=0}^{i-1}\sum_{x_2=0}^{i-x_1}\sum_{x_3=0}^{i+1-x_1-x_2}\cdots\sum_{x_L=0}^{y_L}\sum_{x_{L+1}=0}^{1}\sum_{x_{L+2}=0}^{2-x_1}\sum_{x_{L+3}=0}^{3-x_1-x_2}\cdots\sum_{x_{j-i-1}=0}^{z_{j-i-1}}$$

An alternate form for C_{ij} is

$$C_{ij} = \tag{A-8}$$

$$\binom{n}{j}\sum_{L=0}^{j-i}\binom{j}{j-i-L}\left[\underset{k=1,2,\ldots,L}{\sum_{x_k=0}^{y_k}}\cdots\underset{k=L+1,L+2,\ldots,j-i-1}{\sum_{x_k=0}^{z_k}}\left[\prod_{k=1}^{L}\binom{i+L-\sum_{m=1}^{k-1}x_m}{x_k}\right]\right.$$

$$\left.\cdot\left[\prod_{k=L+1}^{j-i-1}\binom{j-i-L-\sum_{m=1}^{k-1}x_m}{x_k}\right](n-j+i)^{j-i-L-\sum_{m=L+1}^{j-i-1}x_m}\right]$$

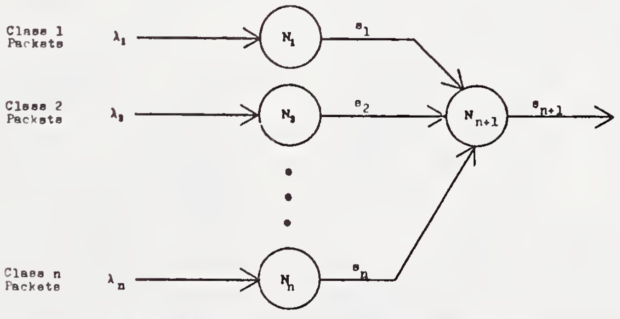


Fig. 1: A Queueing Network Consisting of "n+1" Single Server Queues and "n" External Inputs with a Merger Node at N_{n+1}:

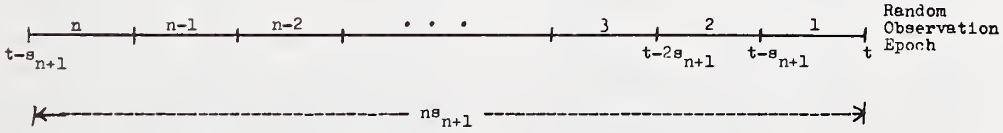


Fig. 2: Illustration of a Random Observation Epoch at the Merger Node of Figure 1.

THE APPLICATION OF MULTIVARIATE STATISTICAL TECHNIQUES TO COMPUTER PERFORMANCE EVALUATION USING SIMULATED DATA

Thomas C. Hartrum

Air Force Institute of Technology
Department of Electrical Engineering
Wright-Patterson AFB, OH 45433

and

Gregory Magavero

Aerospace Guidance and Metrology Center
Newark Air Force Station, OH 43055

This paper considers the application of multivariate statistical techniques to the analysis of data for use in computer performance evaluation (CPE). Traditionally, multilinear regression analysis has been used in the analysis of CPE data. More recently cluster analysis has found applications in this field, both in workload analysis [1] and in performance modeling [2]. However, both approaches have problems when applied to the type of data often encountered in CPE studies. In recent years, several new statistical techniques have been developed to overcome or compensate for these problems. Although these techniques have been used in a variety of business and social applications, little has been reported on their applicability to CPE. This study considered a total of seven multivariate statistical techniques: multilinear regression (as a baseline technique), cluster analysis, ridge regression, automatic interaction detection (AID), canonical correlation analysis, factor analysis, and discriminant analysis. Each technique was examined for its theoretical capabilities and expected usefulness in a CPE environment. Then each technique was applied to several sets of CPE data. In order to conduct a controlled experiment, the data was generated by CPESIM, a simulation of a multiuser mainframe computer system [3]. Based on the results of their application to simulated CPE data, it was found that all of the techniques could be useful to varying degrees and in varying ways [4]. Some gave better predictability than regression, based on the r-squared value, some gave a better analysis of interrelationships between parameters, and so forth. Because of their individual natures, then, each technique was found to be most useful to particular types of problems. Thus a given CPE problem might not be able to use all of the techniques. As a whole, however, these techniques should be considered by anyone performing statistical analysis of computer system data.

Key Words: Automatic interaction detection (AID); canonical correlation analysis; cluster analysis; computer performance analysis; computer performance evaluation; CPE; discriminant analysis; empirical modeling; factor analysis; multilinear regression analysis; ridge regression; statistical techniques.

1. Introduction

There has been considerable interest over the years in modeling computer systems for computer performance evaluation (CPE) purposes. Three types of models are usually recognized: simulation models; analytical models (such as queueing); and empirical models. The use of these models usually falls into one of two catagories. These are for predictive purposes (to pose "what if" type of questions for planning) and for explanatory purposes (to gain a better understanding of how the system works).

The two types of empirical models classically used in CPE are multilinear regression analysis and cluster analysis. Typically regression analysis has been used for performance modeling (e.g. modeling turnaround time as a function of several workload parameters) [1][1], while cluster analysis has been used for modeling workloads [2], although an extension of clustering to performance modeling has been suggested [3]. However, several other statistical techniques have become available in recent years. These have found useful applications in the areas of operations research and in the social sciences. Little, however, has been reported as to their applicability in the area of computer performance evaluation.

This study considers the application of five new techniques to CPE use, along with multilinear regression analysis and cluster analysis as bases for comparison [4],[5]. The five techniques considered are ridge regression, Automatic Interaction Detection (AID), canonical correlation analysis, factor analysis, and discriminant analysis. The objective of this paper is not to provide a theoretical foundation for these techniques (although appropriate references are provided for the interested reader), but rather to present an overview of the techniques and the result of the application of these techniques in a CPE environment.

The basic shortcomings of regression analysis and cluster analysis, along with a brief introduction to the five new techniques, are provided in the Section 2. Then the CPE environment in which this study was made is discussed in Section 3. In Section 4, each of the techniques is presented, along with a discussion of its results in the experiment. Finally, a summary of the results and associated conclusions is presented.

2. Overview of the Techniques

Multilinear regression analysis was probably the earliest used empirical modeling technique in CPE. The basic approach of regression analysis is to model a dependent variable as a function of several independent (or predictor) variables, on the assumption that this represents a cause and effect relationship. In its most popular form, multiple independent variables are allowed, and the dependent variable is expressed as a linear combination of the independent variables, with the coefficients being the primary model parameters. In an empirical technique a black-box approach is taken. Rather than start with an understanding of how the system works, a model is assumed (e.g. the linear relationship just described) and the model parameters are evaluated from empirical data. This is accomplished by observing the values of the dependent and independent variables in the real system being modeled, then finding a "best" fit of the model to this empirical data.

Several statistical measures can be calculated to determine how well the model fits the data. Perhaps the most important is the coefficient of determination, often referred to as the r-squared value. This number, (between 0 and 1.0) indicates the proportion of variation of the dependent variable about its mean value that can be explained by the model. Thus a model of computer turnaround time with an r-squared value of .82 explains 82% of the variation in turnaround. Although an r-squared value of 1.0 would be

[1] Figures in brackets indicate the literature references at the end of this paper.

ideal, due to the stochastic nature of CPE data such a value is not to be expected. An r-squared greater than measure of the model itself is the statistical significance of each of the coefficients. This provides a means of testing the hypothesis that a given coefficient is zero. If that hypothesis cannot be rejected, then the coefficient can be considered statistically zero, which means that the corresponding independent variable does not influence the dependent variable, and thus can be eliminated from the model. This is an important tool when using the model in an explanatory mode, to understand how the system works.

There are many restrictive assumptions made about the data when performing a multilinear regression analysis. The most important is that the relationship is linear. In many real cases it is not. However, the ability to detect nonlinearity and the formulation of an appropriate nonlinear regression model are not simple, and are beyond the experience of many practical CPE analysts. Another problem that is common in a CPE environment is that the independent variables are not truely independent from each other, but rather may be highly correlated with each other. This has the result of distorting the coefficients, and can lead to a variable being dropped from the model as unimportant when in fact that is not the case. If such a high correlation does exist between variables, the data is said to be multicolinear. The predictive capability of the resulting regression model is often satisfactory, but from an explanatory point of view the importance of predictor variables may be misrepresented in the model.

Ridge regression attempts to reduce the effects of multicolinearity in a regression model [4][5]. As such, it is basically a variation of multilinear regression analysis. A small amount of bias is injected into the calculations in order to generate a set of coefficients that compensate for the effect of multicolinearity.

Factor analysis carries this approach one step further by trying to find a surrogate variable or "factor" to replace the two or more variables that show high correlation [6]. Thus in the previous example a single factor "job size" might replace the correlated variables memory size and CPU time.

Although originally used as a tool to help characterize workloads for generating simulation models, cluster analysis can help detect multicolinearity. Thus, for example, one might find high correlation between memory requirements and CPU time. This might show up as a cluster of jobs with small memory requirements and small CPU requirements, and another cluster of jobs requiring both large amounts of memory and large CPU times. In addition, cluster analysis is useful even when the relationship between predictor variables is nonlinear.

Discriminant analysis is similar to cluster analysis in that one tries to determine a discriminant function which can then be used to assign jobs to different groups [7]. Data known to be in separate groups is first analyzed to determine the appropriate discriminant function. The resulting function can then be used to classify new jobs into the appropriate group. This technique requires that the groups be known in order that the discriminant function can be determined, while traditional clustering algorithms determine the groups by iterative techniques.

Canonical correlation analysis is similar to factor analysis in that it attempts to reduce the dimensionality of the data. It relates one group of variables to a second group of variables. This implies a data set with multiple dependent variables. An example might include turnaround time, I/O time required, and input queue time as dependent variables, with CPU time, number of disk I/Os, number of tape I/Os, and number of lines printed as independent variables.

Another problem with multilinear regression analysis occurs when the value of one variable changes the coefficient (i.e. the effect) of another variable. For example, as a computer job's memory requirement gets larger, the effect of that job's CPU time on its turnaround time may get smaller (perhaps because fewer jobs can fit into the system, hence the job gets more frequent access to the CPU). Thus the coefficient of CPU time for overall jobs isn't really a constant, although regression analysis would assign one. Such data is said to be nonadditive. This condition affects both the predictive and explanatory power of the model, although the former can be corrected for, once nonadditivity has been detected.

Automatic Interaction Detection (AID) is useful in detecting nonadditivity [8]. It is similar to cluster analysis in that it attempts to partition the data into separate groups. However, whereas cluster analysis looks only at the independent variables to form clusters, AID uses predefined boundaries in the dependent variable to group the data. If nonadditivity is detected, it can be corrected for in the regression model by introducing a crossproduct term between the variables in question.

Clearly there are some differences in the types of data or situation in which these different techniques might be applied. It is unlikely that all techniques will be useful in a given problem. In order to compare their usefulness, however, an attempt was made to apply them in as similar an environment as possible. This environment is discussed in the next section.

3. The Experimental Environment

In order to test the various statistical techniques under consideration, a source of computer system performance data was needed. However, as noted in the last section, not all of the techniques were expected to be equally applicable to a given problem. On the other hand, if significantly different systems were used for the tests, then comparison of the results would be less meaningful. Therefore, it was decided that for the initial evaluation of these techniques simulated data would be used. Due to its ready availability, user familiarity, and the fact that it was developed for CPE experiments, the CPESIM simulation was selected.

The CPESIM simulation was developed at the Air Force Institute of Technology in 1979 as a teaching aid for the CPE course taught there [9]. CPESIM is a simulation of a single processor, multiprogrammed, batch oriented mainframe. The peripheral and operating system configurations and the workload can be controlled. The simulated computer processes the simulated workload, and generates accounting data, software monitor data, and hardware monitor data. Thus the use of this simulation allows data to be generated under controlled and variable conditions.

The CPESIM accounting data was used as the source of the data for this study. Table 1 shows the variables used. Turnaround time was used as the dependent variable in all cases, and I/O time was also used as a dependent variable in the canonical correlation analysis.

Table 1. CPESIM Independent Variables (Defined for Each Computer Job)

Variable	Description
CPU	CPU seconds used
MEMORY	K-bytes of CM used
CARDS	Number of cards read
LINES	Number of lines printed
DISKIO	Number of disk accesses
TAPEIO	Number of tape accesses
TAPES	Number of tapes mounted
TARRIV	Job arrival time (in hours)
IOTIME	Total job I/O time (seconds) Highly correlated with DISKIO & TAPEIO
TURN	A job's batch turnaround time

Two primary data sets and two variants were created using CPESIM. These are defined in Table 2. Not all of these sets were used for every technique. In addition, Data set 3 was created for use in demonstrating ridge regression. This latter set was not generated by CPESIM, but was hand-built.

4. Test Results

Multilinear regression analysis was used to model turnaround time as a function of the independent variables. This provided a baseline for comparison of the other techniques. The regression results are indicated in Table 3. The multicolinearity index was generated using the ridge regression technique, as presented later in this paper. Note that Data Set 1, which excludes IOTIME, exhibits little multicolinearity, while

Table 2 - CPESIM Data Sets

Data Set	Observations	Description
1	787	Complex workload from four organizations, each different, over five 8-hour days. High degree of competition for resources. IOTIME not included.
1a	787	Same as Set 1 except IOTIME is used as an idependent variable, introducing known multicolinearity.
2	106	Simplest data set from one organization over 4 days. Virtually no competition for resources. IOTIME is included.
2a	452	Same as Set 2 except extended to twenty days data.
3	25	This set consists of three variables. X1 and X2 are statistically independent, while X2 and X3 are highly correlated.

Data Sets 1a and 2 show increased levels of multicolinearity due to the high correlation of IOTIME with TAPEIO and DISKIO. This effect is diminished in Data Set 1a due to the complexity of the system and the high competition for resources. Data Set 3 was regressed both without X3 and with X3 included. Note that the coefficients of X2 and X3 have a compensating effect, resulting in a predictive capability equivalent to that obtained when X3 was excluded from the model.

4.1 Ridge Regression

The first new technique considered was ridge regression.The objective of this technique is to detect the presence of multicolinearity and to correct for it by generating an improved set of regression coefficients. There is no intent to change the predictive capability of the model. The benefit gained is a set of coefficients that more truely represent the importance of the predictor variables. The ridge regression program generates the new coefficients and also a Variance Inflation Factor (VIF) which is an index of the amount of multicolinearity. Thus little change would be expected for Data Set 1, some change for Data Set 1a, and noticable change for Data Set 2.

Table 3 - Multilinear Regression Results
(Standardized Coefficients)

Data Set	CPU Time	Central Memory	Tape Drives	Lines Prntd	Disk I/Os	Tape I/Os	I/O Time	R-Sqrd	Multicolinearity Index
1	.123	.359	.223	.046	.156	.097	....	.424	1.2
1a	.112	.371	.235	.025	.053	-.132	.265		8.3

Data Set	Tape Drives	Disk I/Os	Tape I/Os	I/O Time	R-Sqrd	Multicolinearity Index
2	-.00094	-.00009	-.00150	.00173	.8334	69.5

Data Set	X1	X2	X3	R-Sqrd	Multicolinearity Index
3	4.07	0.34	....		
3	3.84	28.68	-28.54	.803	2458.9

The approach is to recalculate the regression coefficients, introducing a small amount of bias k into the calculations. When k=0, the same values are generated as for multilinear regression. It is desirable to keep k as small as possible. The algorithm increases k iteratively until some heuristic stopping criterion is met. Those criteria considered in this effort include (1) all VIFs less than 10 [5]; (2) all VIFS less than 1 [4]; and (3) all signs correct [5]. Data Set 3 is used to illustrate the effect. In this set X1 and X2 are statistically independent while X2 and X3 are almost perfectly correlated.

The results are shown in Table 4. For Data Set 3, two of the heuristics resulted in nearly the same coefficients as were generated by multilinear regression when X3 was not included. For Data Set 2, the coefficients changed significantly with little degradation in r-squared. For Data Set 1a, there was a noticeable, but small change in the coefficients. Again r-squared was not affected. Finally, as expected, for Data Set 1 two of the heuristics indicated that no bias should be added at all. The criterion "all VIFs less than 1" did require a bias of k=0.105, but the actual coefficients changed little.

Table 4. Ridge Regression Results (Standardized Coefficients).

Data Set 1 (I/O Time Excluded.)

k	CPU Time	Central Memory	Tape Drives	Lines Prntd	Disk I/Os	Tape I/Os	I/O Time	Heuristic
0.000	.123	.359	.223	.046	.156	.097	----	VIFs<10, Signs
0.105	.118	.329	.211	.053	.145	.098	----	VIFs < 1

Data Set 1a (I/O Time Included.)

k	CPU Time	Central Memory	Tape Drives	Lines Prntd	Disk I/Os	Tape I/Os	I/O Time	Heuristic
0.000	.112	.371	.235	.025	.053	-.132	.265	OLS
0.015	.111	.365	.231	.030	.084	-.059	.181	VIFS < 10
0.070	.109	.347	.222	.037	.105	.002	.117	Signs
0.110	.108	.335	.216	.040	.107	.015	.105	VIFs < 1

Data Set 2

k	CPU Time	Central Memory	Tape Drives	Lines Prntd	Disk I/Os	Tape I/Os	I/O Time	Heuristic	R-sqr
0.000			-.00094		-.00009	-.00150	.00173	OLS	.8334
0.012			-.00133		.00001	-.00025	.00051	VIFs<10	.7553
0.044			-.00124		.00000	.00000	.00026	Signs	.7199
0.092			-.00102		.00001	.00006	.00019	VIFs<1	.7093

Data Set 3

k	X1	X2	X3	Heuristic	R-sqr
0.000	3.84	28.7	-28.5	OLS	.8030
0.004	4.05	1.12	-0.79	VIFs < 10	.7932
0.024	3.97	0.34	-0.01	VIFs < 1	
0.026	3.97	0.33	0.00	Signs correct	.7922

Overall, ridge regression did improve the regression coefficients, most notably in those cases exhibiting large multicolinearity. The criteria "all VIFs less than 1" and "all signs correct" gave approximately the same results, and appeared to give better results than "all VIFs less than 10".

4.2 Factor Analysis

Factor analysis provides another solution to the problem of multicolinearity. The objective here is to find the underlying structure of the data, then generate a new set of variables, called "factors," to describe the data. Usually there are fewer factors than original predictor variables, and little or no multicolinearity should exist among them. Thus for Data Set 3, two factors would be generated. One would be equivalent to X1, while the other would be representative of X2 and X3.

This technique yields results that may be more useful from an explanatory point of view than ridge regression. The latter may still tend to hide the importance of a variable. Consider Data Set 3. One could argue that both X2 and X3 are equally important, and either could be used for predictive purposes. While ridge regression reduced the inflated coefficient of X2 to reflect its proper importance, it essentially eliminated X3 by giving it a near zero coefficient. Factor analysis would explicitly show that there were two underlying factors that characterize the data, and that both X2 and X3 were equally related to the second factor. One then usually has two options for replacing predictor variables with factors. One approach is to use surrogate variables. Here a predictor variable is found that has a high correlation with the factor (called a "high loading"), and that variable is used. In Data Set 3, such an approach would indicate the use of X1 and X2 (or equally X1 and X3) as the surrogate variables. If no single variable is highly loaded on a factor, then the factor values may be calculated as a weighted (by the loadings) sum of the predictor values. Once one of these techniques has been applied, then multilinear regression is applied to the factors to generate the new model. It should be noted that factor analysis is applied to only the predictor variables; the dependent variable is not included in the factor analysis itself.

This technique was applied to Data Sets 1, 1a, and 2a. The results are shown in Table 5. In all cases only two underlying factors could be found. The results for Data Set 1a are very appealing. IOTIME, DISKIO, and TAPEIO were highly loaded on one factor, while everything else was moderately loaded on the other factor. A good approach might be to replace IOTIME, DISKIO, and TAPEIO with Factor One, and leave the other variables alone. 57% of the variation in the predictor variables is accounted for by the factors for this data set. Similarly, the results for Data Set 2a indicates one factor represents disk I/O (which seems to account for most of the I/O time) while the other factor represents tape I/O. The high correlations result in these factors accounting for 89% of the predictor variables. The results for Data Set 1 are more difficult to interpret. Factor one is a combination of MEMORY, TAPES, LINES, and TAPEIO, while factor two seems to depend primarily on TARRIV and CARDS. Only 43% of the variation in the predictor variables is modeled by these factors.

Table 5. Factor Analysis Results (Varimax Rotated Factor Matrix After Rotation with Kaiser Normalization)

Data Set	Variable	Principal Components Factor Number 1	2
1	TARRIV	.1350	-.6048
	CPU	.5659	.2576
	MEMORY	.6370	-.3271
	TAPES	.7529	.0738
	CARDS	.2642	.6574
	LINES	.5887	.0785
	DISKIO	.3402	.4274
	TAPEIO	.6297	.1912
1a	IOTIME	.9553	.2357
	CPU	.1737	.6184
	MEMORY	.0189	.6904
	TAPES	.3825	.6244
	LINES	.1280	.6591
	DISKIO	.6140	.0529
	TAPEIO	.8411	.2412
2a	IOTIME	.9469	.3111
	TAPES	.0074	.8883
	DISKIO	.9876	.0349
	TAPEIO	.2140	.8694

When highly related predictor variables are in the model, factor analysis would seem to do a better job of eliminating multicolinearity than ridge regression. However, it is a more involved procedure.

4.3 Cluster Analysis

Cluster analysis provides another method for attacking the problem of some underlying relationship between the supposedly independent predictor variables. Multicolinearity implies a linear relationship between the predictor variables. Thus a scattergram of two highly correlated variables would resemble a straight line.

Consider a nonlinear example. Suppose that all jobs that require small amounts of memory use large amounts of CPU time. In addition, all jobs that require large amounts of memory require large amounts of CPU time. All jobs with small CPU requirements use an intermediate amount of memory. A scattergram for such a workload might appear as in Figure 1. CPU time and memory are not highly correlated, nor would factor analysis find a single underlying factor to replace both of them. Yet clearly the two are not truely independent.

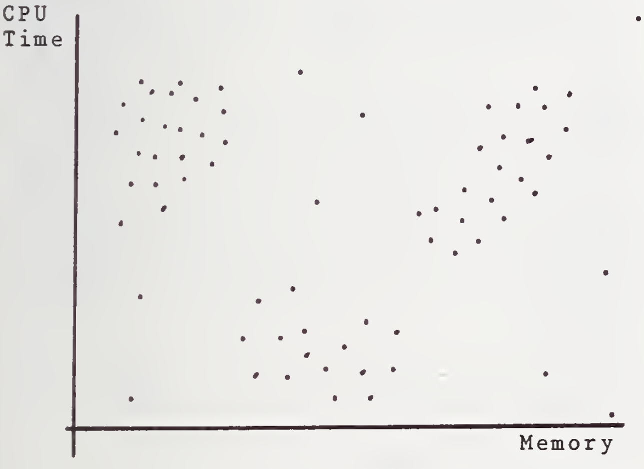


Figure 1. Nonlinear Relationship of Data

Cluster analysis approaches this problem by partitioning the workload into clusters, or groups, of similar jobs, based on the workload parameters (i.e. the predictor variables). Then a separate regression model can be developed for each cluster. Cluster analysis starts with no knowledge of such clusters, and then iteratively converges to a set of natural clusters.

When cluster analysis was applied to the data sets in question, natural clusters were found. Much of the workload was discarded as falling between clusters. No improvement to the turnaround time model was found using the resulting clusters. This is primarily a result of the fact that CPESIM currently generates the workload parameters independently, and does not provide for producing related parameters. However, previous studies [3] have indicated a usefulness of cluster analysis to this problem. In addition, cluster analysis has been well accepted by the CPE community for workload analysis and generation of simulation inputs [2].

4.4 Discriminant Analysis

Once it is known that jobs can be catagorized into discrete groups, discriminant analysis provides a technique for classifying them. The groups must be known in advance. Discriminant analysis determines a discriminant function, based on empirical data, that can then be used to determine the proper group for new jobs. Some possible applications to CPE data are such things as (1) to which user organization does each job belong?; (2) to which cluster does a job belong (but this can be done using cluster analysis)?: or (3) to which class of turnaround times will a job belong? This latter question might be useful in predicting turnaround times. A problem is that discriminant analysis works best when there are clearly defined, distinct groups. Turnaround time tends to be a continuously distributed variable. Thus arbitrarily partitioning turnaround time will tend to generate groups that are not well separated.

This technique was applied to Data Set 1. For the first test, turnaround time was partitioned into three classes: jobs with turnaround times of less than 0.5 hour; those with turnaround times between 0.5 and 1.5 hours; and those jobs with turnaround times greater than 1.5 hours. Half of the 787 cases were randomly selected and used to generate the discriminant function, the other half were used to test it. Table 6 shows the resulting three group centroids. Table 7 shows the resulting classification errors.

In an attempt to find more clearly separable groups, the data was divided into two groups at a turnaround time of 3.2 hours. These represented "regular" jobs and slow jobs. However, only nine

Table 6. Group Centroids

Group Number	Discriminant Function
1	1.226
2	-0.191
3	-1.086

Table 7. Classification Matrices for Discriminant Analysis Example, 3 Groups

Results for Cases Used for Analysis

Actual Group	Number of Cases	Predicted Group 1	2	3
1	107	85 79.4%	18 16.8%	4 3.7%
2	186	48 25.8%	84 45.2%	54 29.0%
3	88	5 5.7%	25 28.4%	58 65.9%

59.58% grouped correctly.

Results for Cases Not Used for Analysis

Actual Group	Number of Cases	Predicted Group 1	2	3
1	121	95 78.5%	23 19.0%	3 2.5%
2	192	53 27.6%	79 41.1%	60 31.1%
3	93	4 4.3%	31 33.3%	58 62.4%

57.14% grouped correctly.

jobs fell into the slow group, leaving 778 cases in the regular group. The two group centroids are shown in Table 8, while the classification errors are indicated in Table 9. Note that 113 of the regular jobs were misclassified as slow jobs - considerably more than the nine actual ones.

Discriminant analysis is not a useful tool unless the data is known to fit into well defined, distinct classes. The CPESIM data did not provide a good example of such an environment.

Table 8. Discriminant Function Coefficients

Group	Discriminant Function
Normal	-0.02472
Outliers	2.13691

Table 9. Classification Matrices for Discriminant Analysis Example, 2 Groups

Classification Results

Actual Group	Number of Cases	Predicted Group Normal	Outliers
Normal	778	665 85.5%	113 14.5%
Outliers	9	1 11.1%	8 88.9%

85.51% of all cases classified correctly

4.5 Canonical Correlation Analysis

Canonical correlation analysis is a technique for reducing the dimensionality of data. It relates two sets, each of which contains several related variables. Similar to regression analysis, which models a single dependent variable as a function of a set of predicitor variables, this technique allows a set of related dependent variables to be modeled as a function of a set of predictor variables. Note that each set should consist of variables that are somehow related.

The approach is to define two canonical variates, X* and Y*, where

$$X^* = \sum AiXi \tag{1}$$

and

$$Y^* = \sum BiYi \tag{2}$$

where the Xi represent the independent, or predictor, variables and the Yi represent the dependent variables. These two models are solved simultaneously for the coefficients subject to maximizing the correlation between X* and Y*. The coefficients indicate the contribution of each variable to the corresponding canonical variate. Canonical loadings measure the correlation of each variable with its canonical variate. Redundancy is defined as a measure of the amount of variaton in the predictor set that is explained by the dependent set, similar to the r-squared value in regression.

This technique was applied to Data Sets 1a and 2a. Turnaround time and IOTIME were included in the dependent set as related variables, and the remaining predictor variables were included in the other set. The results for Data Set 1 are shown in Table 10. For the y variate, note the high loading of IOTIME for one solution and for turnaround time for the other solution. This implies that these two dependent variables are not really related. Note also the loadings of the X variates. For this case, two independent regression models would have been more appropriate. Table 11 presents the results for Data Set 2a. These results are more useful. Note that both dependent variables are highly loaded on the canonical variate in the first solution. Notice the high redundancy. The X loadings indicate the importance of the individual predictor variables in determining the dependent set, which might be characterized as "job run time" in this case.

Thus canonical correlaton has limited usefulness in CPE. The primary problem with its application is defining a set of related dependent variables of interest.

4.6 Automatic Interaction Detection (AID)

Automatic interaction detection (AID) is another data partitioning scheme, but with a different objective. Its purpose is to detect the presence of nonadditivity, or interaction between variables. The data is split into two groups based on one of the predictor variables in order to form the two groups with the least within-group variation of the dependent variable values and the maximum between-groups variation. Then each group is further split and so forth, creating a tree. If the resulting tree is symmetrical, then no interaction between variables is suspected. A nonsymmetric tree indicates that interaction exists. Such nonadditivity can be corrected for (in a predictive sense) by adding product terms to the regression model.

This technique was applied to Data Sets 1 and 2. Figure 2 shows the resulting tree for Data Set 2. Since this was the simple data set, intuitively no interaction was expected, and the symmetric tree bears that out. Figure 3 is the tree for Data Set 1. Here assymetry is cleary

Table 10. Results of Canonical Correlation Analysis (Data Set 1a)

Variate	CANVAR1		CANVAR2		
	Canonical		Canonical		
x-variables	Coeff	Loading	Coeff	Loading	Total
TARRIV	0.0070	-0.0353	0.4777	0.4997	
CPU	0.0105	0.2998	0.0845	0.2573	
MEMORY	0.0068	0.2334	0.3789	0.5366	
DISKIO	0.2157	0.3842	-0.0695	0.0415	
TAPEIO	0.9290	0.9755	-0.4147	-0.0224	
TAPES	-0.0019	0.4612	0.7743	0.6981	
CARDS	0.0061	0.2566	-0.0179	0.0350	
LINES	0.0203	0.2743	-0.0662	0.1526	
Redundancy		0.1972		0.0798	0.2770
x-variables					
TURN	0.0026	0.3713	1.0760	0.9285	
IOTIME	0.9990	1.0000	-0.3995	-0.0024	
Redundancy		0.5615		0.2462	0.8077

Table 11. Results of Canonical Correlation Analysis
(Data Set 2a)

Variate	CANVAR1		CANVAR2		
	Canonical		Canonical		
x-variables	Coeff	Loading	Coeff	Loading	Total
TARRIV	-0.0004	0.0126	-0.0077	-0.0949	
CPU	0.0040	-0.0104	-0.6379	-0.6499	
MEMORY	-0.0034	0.0054	-0.0350	-0.0454	
DISKIO	0.8702	0.9183	0.0474	0.0154	
TAPEIO	0.3932	0.4988	0.0752	0.0675	
TAPES	-0.0011	0.2606	-0.0158	0.0054	
CARDS	0.0112	0.0788	-0.3000	-0.3323	
LINES	0.0490	0.0863	-0.6908	-0.6916	
Redundancy		0.1453		0.1057	0.2510
x-variables					
TURN	-0.0369	0.7969	-1.7040	-0.6041	
IOTIME	1.0297	0.9998	1.3582	-0.0217	
Redundancy		0.8095		0.1504	0.9599

evident, indicating that interaction is present. Adding the two product terms (MEMORY)x(TARRIV) and (DRIVES)x(TARRIV) improved the r-squared value from 0.424 to 0.505. These terms make sense in light of the fact that other techniques have indicated a problem due to tape drives and accesses, and as the system starts to backlog, jobs with a later arrival time will tend to be impacted even more.

Thus automatic interaction detection does seem to be a good tool for the detection of interaction (nonadditivity) between variables in CPE modeling.

5. Conclusions

This study considered the application of seven techniques to the empirical modeling of turnaround time in a CPE environment. Multiple linear regression analysis was used as a baseline on four sets of data. Then each of the other techniques was applied to see if the model could be improved. It was found that ridge regression was a useful tool in removing the distortion of regression coefficients due to multicolinearity, but that it tended to give the proper coefficient for one variable at the expense of the related variable. Factor analysis, on the other hand, took a more basic approach by finding a new set of variables, or factors, to characterize the workload. The relation of multicolinear variables to these factors could be clearly seen, and multicolinearity was removed from the model when the factors were used in place of the original predictor variables. Cluster analysis extended this solution to cases where a nonlinear relationship exists between predictor variables by partitioning the workload into sets of similar jobs, so that when modeled separately multicolinearity was not a problem. Discriminant analysis appears to have limited use in CPE, providing a technique for classifying jobs into previously defined, clearly distinct groups. Similarly, canonical correlation analysis seems to be useful only when a group of dependent performance variables can by defined to be modeled as a function of the predictor variables. Finally, Automatic Interaction Detection (AID) was found to be a useful tool for the detection of nonadditivity, or interaction, between variables.

Current efforts are under way to test these techniques further. Data is being generated to allow explicit demonstration of the capabilities of cluster analysis, and to demonstrate the ability to use several of these techniques jointly to improve a given regression model. In addition to these tests with simulated data, an effort is under way to apply these techniques to data measured on an actual online system.

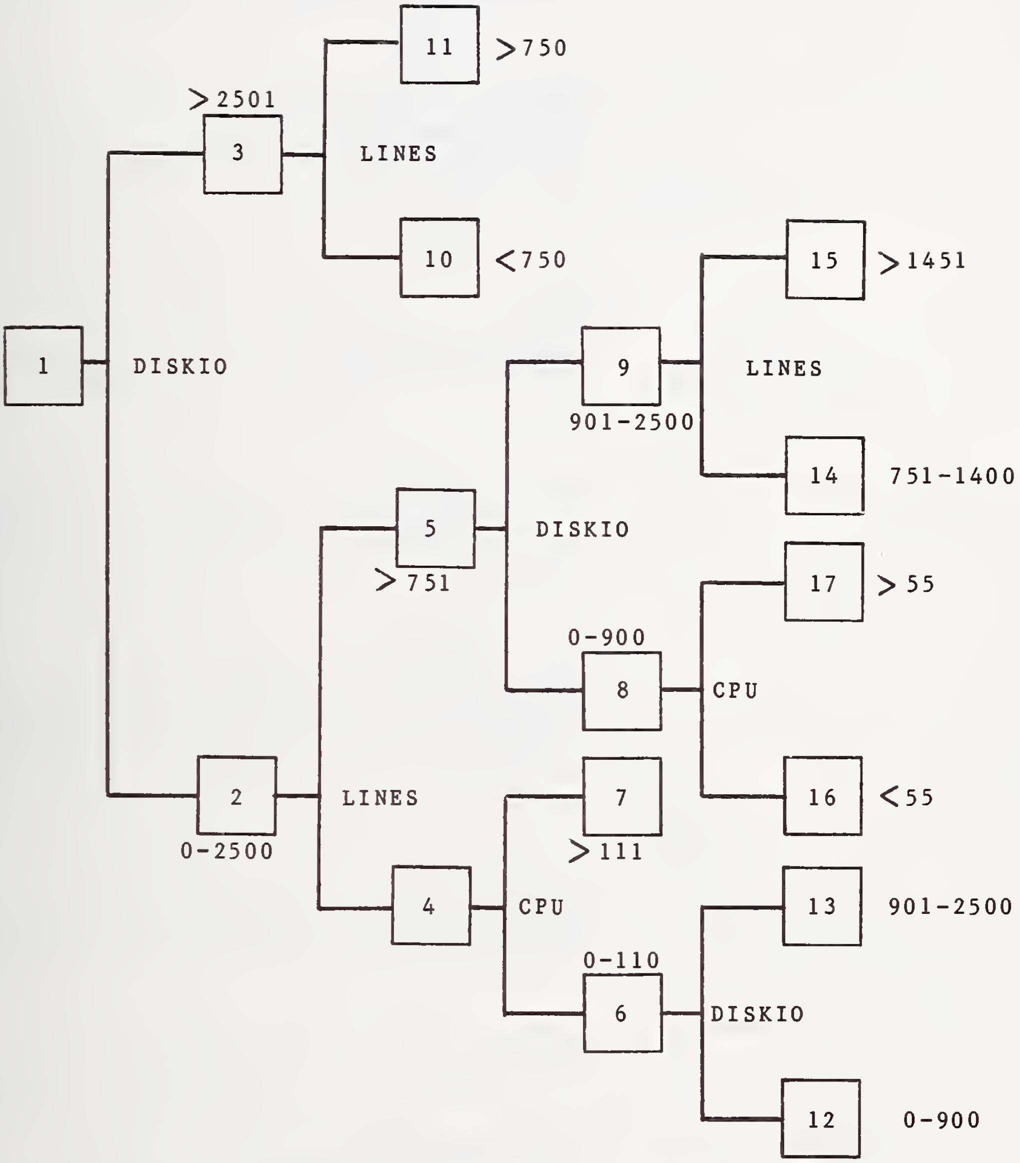


Figure 2. Augmented AID Skeleton Tree for Data Set 2a

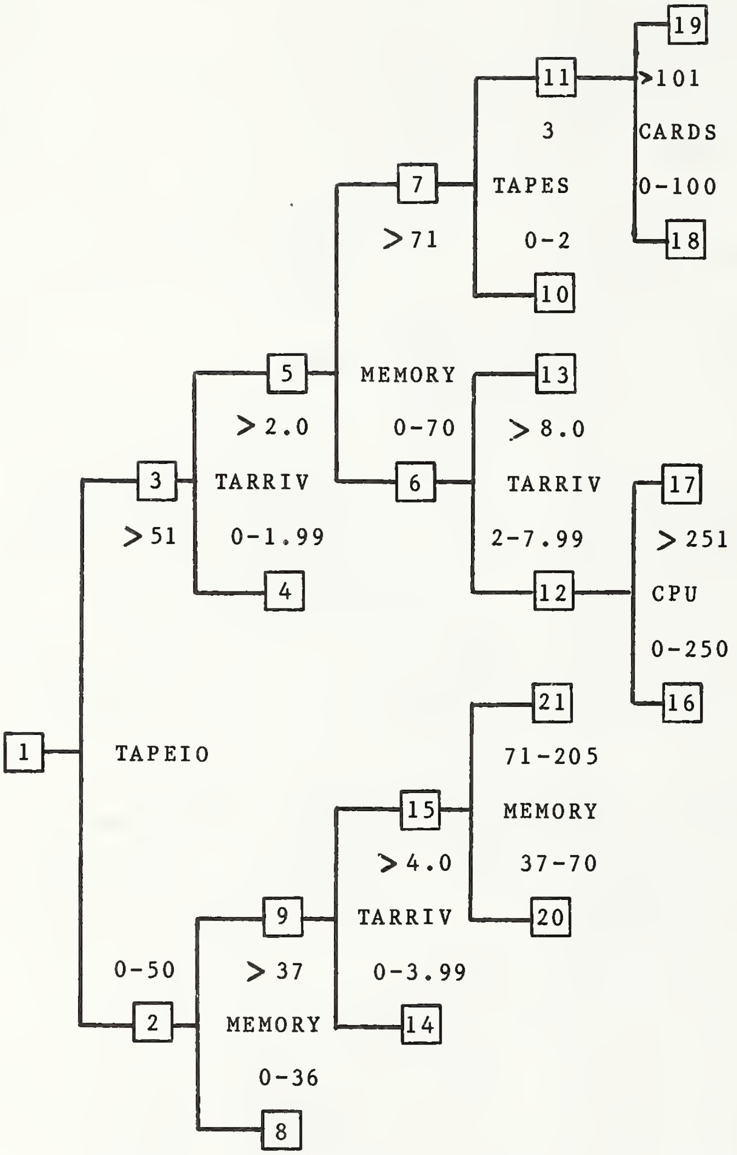


Figure 3. Augmented AID Skeleton Tree for Data Set 1

References

[1] Gomaa, H. "A Modelling Approach to the Evaluation of Computer System Performance," pp. 171-179, Modelling and Performance Evaluation of Computer Systems, North Holland Publishing Company, 1976.

[2] Agrawala, A.K. and J.M. Mohr, "Some Results of the Clustering Approach to Workload Modelling," Proceedings of the Thirteenth Meeting of the Computer Performance Evaluation Users Group, pp. 23-38, 1977.

[3] Hartrum, Thomas C. and Jimmy W. Thompson, "The Application of Clustering Techniques to Computer Performance Modelling," Proceedings of the 15th Meeting of the Computer Performance User's Group, October, 1979.

[4] Stover, Margaret A., Application of Statistical Analysis Techniques to Computer Performance Evaluation, Master's Thesis, Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio, 1981.

[5] Magavero, Gregory, A Study of Multivariate Statistical Analysis Techniques for Computer Performance Evaluation, Master's Thesis, Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio, AFIT/GCS/EE/82D-23, 1982.

[6] Bennett, Spencer and Davey Bowers, An Introduction to Multivariate Techniques for Social and Behavioral Sciences, John Wiley and Sons, New York, 1976.

[7] Nie, Norman H., et al, SPSS Statistical Package for the Social Sciences, McGraw-Hill, N.Y., 1975.

[8] Gooch, L.L., Policy Capturing with Local Models: the Application of the Automatic Interaction Detection Technique in Modeling Judgement, unpublished PhD dissertation, University of Texas, Austin, TX, 1972.

[9] Hartrum, Thomas C., "CPESIM - A Teaching Tool for CPE," Annual Computer Performance Management Users' Group Meeting - Minutes, Apr. 14-17, 1980, Gunter AFS, AL.

IMPROVING THE ACCURACY OF A WORKING-SET-ORIENTED GENERATIVE MODEL OF PROGRAM BEHAVIOR[1]

Domenico Ferrari
Tzong-yu Paul Lee

Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA 94720

An experiment based on trace-driven simulation is carried out to study various improvement strategies for a working-set-oriented generative model. Working set size strings extracted from a real program trace are used as inputs to the generative model. The memory demand and page fault rate of these artificial memory reference strings are compared with those of the original real reference string under the working set policy. The same strings are also tested under two different memory management policies : the page fault frequency policy and the least recently used policy. Artificial strings generated with proper strategies behave quite well under both the working set and the page fault frequency policies. However, they behave less than satisfactorily under the least recently used policy.

Key words: Generative model; program behavior; working set policy; workload characterization.

1. Introduction

In the study of program behavior, memory referencing patterns have long been a subject of interest. On the one hand, various memory management policies have been proposed and implemented to optimize the utilization of system resources and increase system performance. On the other hand, techniques that enable compilers to generate better-behaved code in order to achieve the same purpose have been studied. Various methods for restructuring programs so as to improve their referencing behavior have also been investigated [Ferr76a].

Very often, in these studies, program reference strings are used to compare the performances of various memory management policies or to validate models of program behavior. These strings are usually collected by interpretively executing real programs on an existing system, but this process is rather tedious and generally very expensive. The use of artificial strings instead of real strings has some clear advantages if the former reflect the behavior of the latter. Artificial strings are usually produced by generative models of program behavior [Sage73a], that ideally should use relatively few parameters and be based on relatively fast algorithms.

It is usually not necessary to reproduce a program's behavior exactly; consequently, the generative model will try to reproduce those properties of a

[1] The research reported here has been supported by the National Science Foundation under grant MCS80-12900.

real string which are deemed to be important in the context in which the artificial string is to be used. In this study, the memory demand and the page fault rate have been chosen as the important aspects of a real string to be reproduced. Our primary context is that of the working set policy [Denn68a], but we shall also be comparing a real and several artificial strings under the page fault frequency policy [Chu76a] and the least recently used policy [Matt70a]. For convenience, these policies shall be referred to as WS, PFF, and LRU respectively.

The generative model evaluated here, which is based on a working set characterization and reproduces given working set size dynamics, was proposed by Ferrari [Ferr81a] and first implemented by Dutt [Dutt81a]. This paper deals with various improvement strategies that can be applied to this working-set-oriented generative model.

The generative model is described in the next section. The proposed improvement strategies for the model and the design of the experiments are detailed in Sections 3 and 4, respectively. The results and their analysis are presented in Section 5. Finally, in the last section, conclusions are drawn from the outcome of the experiment, and directions for future research are provided.

2. Background

The goal of a generative model is to provide a sequence of page references whose characteristics of interest are similar to those of the modeled program. The generative model evaluated here is based on the sequence (or on sequences) of working set sizes and possibly on other parameters. While it is desirable to reproduce the dynamic behavior of the modeled program as accurately as possible, it is also important to minimize the number of parameters needed for the generative model. A sequence of working set sizes is generally redundant in its information content, as it can be completely specified by the sequence of the pairs (t_i, w_i), where t_i is the time at which the working set size (wss) curve changes its slope, and w_i is the wss at that time. This sequence is the input to the generative model used in the experiment.

When a sequence of wss's is given, the window size T used to define the working sets must always be specified. The reference string generated by the model on the basis of the given wss curve does not coincide with the original real string (if there is one). However, with appropriate generation algorithms, the generated string might be made to possess the essential characteristics of the modeled one. In principle, all possible reference strings of length n form a vector space S^n where S is the page set. By specifying a particular sequence of wss's, we focus on the subspace of this vector space whose elements all have the given wss characterization.

The input to the generative model is the given wss sequence, which is defined for a given window size T. The wss sequence corresponding to the artificial string produced by the model with a different window size will generally be different from that of the modeled program under the same conditions. Performance indices of other kinds are usually different also.

The string generation algorithms used in this study can be classified into two general categories : the *single T* generation algorithms and the *double T* algorithms. By 'single T generation' we refer to the generation of an artificial string based on one wss sequence, corresponding to one value of T. By 'double T generation' we refer to the generation of an artificial string based on two wss sequences. By specifying two wss sequences with two different window sizes, the subset of reference strings that can be generated is further constrained. With the use of appropriate generation strategies, the artificial string is expected to represent the modeled program's behavior under various memory management policies more accurately than the string generated by a single T wss characterization.

The properties of working set size strings and the feasibility of both single and double T generation algorithms are treated extensively in [Ferr81b], [Ferr82a], and [Lee82b].

3. Improvement Strategies

The original generative model incorporated a very simple strategy, since its main purpose was to gain some insight into the behavior of artificial reference strings [Dutt81a]. The artificial reference string was generated with a single working set size string extracted from a real program reference string. In the strategy, three queues of pages are maintained : the *external* queue (E), the *candidate* queue (C), and the *forbidden* queue (F). Initially, all pages are in the external queue. Whenever the working set size increases, a new page is chosen from the external queue and used as the page to be referenced next. When the working set size does not increase, the next reference is chosen from the candidate queue. Now, if the working set size does not decrease T units of time later, the chosen page is put into the candidate queue, since it is to be re-referenced within the next working set window. Otherwise, the page is put into the forbidden queue, whose members cannot be referenced. Furthermore, if the working set size decreases, the page in the forbidden queue that was referenced T time units earlier is moved to the external queue for future use. The candidate queue is managed by a FIFO policy for simplicity, and the external queue by a LIFO policy for economy.

With these simple data structures and strategies, the artificial string generated is fairly accurate under the WS policy with a window size larger than or equal to the one used in its generation. However, its behavior under both the PFF and LRU policies is unacceptable [Dutt81a]. Under the PFF policy, once enough pages manage to be in memory, no further page fault can ever occur. Under the LRU policy, the cyclic referencing behavior due to the FIFO management of the candidate queue creates a large number of page faults. Based on these observations, three general improvement strategies were proposed.

(1) Use Two Working Set Size Strings

The idea behind this approach is that forcing the artificial string to follow two dynamic wss characterizations with two reasonably spaced working set window sizes may coerce it to reproduce the original string's behavior more accurately. For example, the artificial string generated with this strategy may not be as sensitive to a change of window size.

(2) Use All Pages

All distinct pages that are referenced in the real program reference string will be put in the external queue and used in some fashion. Instead of managing the external queue by a LIFO policy, new pages are chosen according to one of two criteria. The first is to cycle the pages in the external queue in FIFO order. Intuitively, each page will be referenced in the artificial string when its turn comes. The second is to use the pages in the external queue in such a way as to match the real program profile, i.e., the relative referencing frequency of each page. The rationale behind the latter approach is to account, at least to some extent, for the identity of the referenced pages; this attempt at reproducing referencing frequencies may improve the behavior of the artificial reference string under various memory management policies.

(3) Reuse Previously Referenced Page

The cyclic referencing pattern of the original strategy causes severe problems with the LRU policy. Program referencing behaviors generally exhibit a high degree of locality. In particular, there is a high probability that the current page coincides with the page just referenced.

4. Design of the Experiments

4.1. Parameters of the model

The parameters of the model are estimated from a real trace. This trace was obtained from the interpretive execution of an APL program on an IBM 360/91 machine. Except for the wss characterizations, which were obtained from the first 550,000 references, all parameters were derived for the first 500,000 references. Various performance indices of the real trace, later used for comparisons, were also gathered by trace-driven simulations from the same 500,000 references.

Three wss characterizations were obtained, with window sizes of 5000, 10000, and 20000 references, respectively. For single T generation, the wss characterization with window size 10000 was used, whereas the other two were used for double T generation. As mentioned in Section 2, a wss characterization is a sequence of (t,w) pairs, where t is a time at which the slope of the wss curve changes, and w is the value of the wss at that time. The numbers of pairs needed by the three characterizations of the APL program trace are 1157, 619, and 525 for window sizes of 5000, 10000, and 20000, respectively. The nominal window size of 10000 was chosen for two reasons. First, this window is not so short as to obliterate the program's phase transition behavior [Dutt81a] and not so long as to require too much memory space. Secondly, in the neighborhood of window size 10000, the space-time product curve shows rather stable and relatively low values.

The *coefficient of resilience* is defined here as the probability that the page referenced next is the same as the currently referenced one. In essence, this is the probability of referencing the top of the stack in an LRU environment (it is often called d_1 in the stack distance probability distribution [Spir77a]). The value of the estimate of this parameter from the real trace is 0.544.

The number of distinct pages in the first 500,000 references of the real trace was found to be 110. Not surprisingly, the most frequently referenced page accounts for 25 percent of the references; also, 20 percent of the pages account for 86 percent of the references.

4.2. Performance indices

Various performance indices were chosen for comparing real and artificial strings. To compute the space-time product, the page wait time was assumed to be constant and equal to 10000 references. The primary performance indices considered in the various contexts are listed below.

(1) WS environment

Mean working set size, page fault rate, space-time product, working set size distribution, and interfault time distribution are the primary indices we are interested in when the WS policy is used.

(2) PFF environment

Mean working set size, page fault rate, space-time product, working set size distribution, and interfault time distribution are the primary indices of concern in the PFF case. The parameter I of the PFF algorithm, i.e., the threshold of interfault times, was chosen to equal 1543 references. This is the value of the mean interfault time obtained under the WS policy with a window size equal to 10000 references.

(3) LRU environment

Page fault rate, space-time product, interfault time distribution, and stack distance probability distribution are the primary indices of concern in the LRU case. For the stack distance probability distribution, the probability d_1 of referencing the top of the stack is particularly important. The parameter of the LRU policy, i.e., the fixed partition size, was chosen to be 21 page frames. This is the mean working set size with a window size of 10000. Under the assumptions made by Denning and Schwartz [Denn72a], this choice for LRU should produce the same page fault rate as the WS policy with window size 10000.

4.3. Overview of the Experiments

4.3.1. Artificial Strings

Twelve artificial strings, each 500,000 references long, were generated and compared with the real string. The names of the artificial strings reflect the three control variables of the experiment : the number of wss characterizations, the way to select a new page when one is needed, and the method for re-using pages already in the working set.

A string name consists of three characters XYZ, where X is S in the case of single T generation, and D in the case of double T generation; the binary variable Y is 0 if the old pages are re-used in FIFO order (i.e., pages are selected from the candidate queue in the order in which they were put in), and is 1 if the probability of referencing the previously referenced page rather than the first in the candidate queue is taken into account (in this case, the number of consecutive references to the same page is geometrically distributed); finally, the variable Z is 0 if new pages are selected from the external queue in LIFO order, 1 if new pages are selected from the external queue in FIFO order (in this case, the external page queue initially contains a number of pages equal to the the total number of distinct pages referenced in the real string), and 2 if new pages are selected from the external queue according to a given relative frequency distribution of page references (in this case, the usage record of each page is continuously updated so that the appropriate page can be chosen to match the given frequency distribution).

4.3.2. Data structures

To generate the six strings named S**, the simple 3-queue data structure described in Section 3 is used.

To generate the six strings named D**, we could operate with two sets of queues, each corresponding to one wss characterization : one set will consist of C_1, F_1, and E_1, and the other of C_2, F_2, and E_2. Each page would be at any given time in one (and only one) of the three queues in each set. However, if these six queues were implemented as described, it would be necessary to calculate a large number of intersections of two queues, one from each set. To speed up the generation algorithm, a data structure consisting of five doubly-linked lists (queues) was implemented. Each queue in the structure corresponds to a particular intersection of the two queues mentioned above. Fortunately, not all possible intersections of the six original queues are needed in the generation of reference strings. The five (intersection) queues required are C_1C_2, F_1F_2, E_1E_2, C_1E_2, and C_1F_2.

All pages are initially in the E_1E_2 queue. When in both wss characterizations the wss increases, a new page is selected from the E_1E_2 queue according to the strategy specified by variable Z. When in both wss characterizations the wss does not increase, a page is selected from the C_1C_2 queue. When the wss with the larger T does not have to be increased and the other needs to be increased, a page is selected from the C_1E_2 queue.

After the page to be referenced next is chosen, the queue to which this page should be added is to be selected. When the page has to be re-referenced in order to remain in both working sets later, the page is put into the C_1C_2 queue. When the page has to drop out of both working sets later, and cannot therefore be re-referenced until then, it is put into the F_1F_2 queue. When the page has to remain in the working set with the larger T, but has to drop out of the working set with the smaller T, the page is put into the C_1F_2 queue.

A page stays in the F_1F_2 queue until it drops out of both working sets, at which time the page is released and moved to the E_1E_2 queue. Similarly, a page stays in the C_1F_2 queue until it drops out of the working set with the smaller T, at which time the page is released and sent to the C_1E_2 queue. No other transitions of pages between queues are possible. It should be noted that this arrangement of the data structures also maintains the chronological ordering of arrival of the pages in each queue. Therefore, no search is needed to select and delete a page when the FIFO strategy is used.

5. Experimental Results and Their Analysis

The generation of 500,000 references with a single window size took from 275 seconds to 421 seconds of VAX-11/780 CPU time depending on the options chosen. The generation of 500,000 references with two window sizes took from 306 seconds to 466 seconds of VAX-11/780 CPU time depending on the options chosen. The remarkable efficiency of the double T generation algorithm is due to the carefully planned structure of the queues, which eliminates the need to do linear searches on them in most cases. An optimized version of the generation program, without statistics gathering, is expected to run twice as fast as the version we have used to gather such voluminous statistics as page reference distribution.

The strings were run under various memory management policies, and their performance indices were compared with those produced under the same policies by the real string. These comparisons are discussed in the following subsections. Notice that string S00 is essentially the same as the string generated in an earlier experiment [Dutt81a]..

5.1. Characterization of Artificial Strings

Two statistics are listed in Table 1 for comparison : the total number of distinct pages used in each string, and the coefficient of resilience. The number of distinct pages used in S02, S12, D02, and D12 would reach 110 if the string generated were infinitely long. However, due to the very small probability densities at the tail of the distribution, only a fraction of all pages are actually used when generating 500,000 references.

5.2. WS policy results

Artificial strings were executed under the WS policy with window size 10000 if they were generated with one T, and also with other window sizes if they were generated with two T's. Their performance indices were found to be exactly the same as those of the real string executed under the same window size(s). This was expected, but strengthened our confidence in the correctness of the generation programs. The results for the real string under the WS policy with several different window sizes are given in Table 2.

Table 1. Characteristics of the Strings

string	number of distinct pages	coefficient of resilience
S00	56	0.000
S01	110	0.000
S02	80	0.000
S10	56	0.544
S11	110	0.544
S12	80	0.544
Real	110	0.544
D00	78	0.000
D01	110	0.000
D02	80	0.000
D10	78	0.544
D11	110	0.544
D12	80	0.544

Table 2. WS Results for the Real String

window size	mean wss	max wss	changes of slope	space time product	page fault rate	max interfault time
20000	26.17	78	525	1.10E8	0.000562	111695
15000	23.67	67	554	1.07E8	0.000592	111695
10000	20.90	56	619	1.06E8	0.000648	111695
7500	19.29	51	742	1.14E8	0.000770	65292
5000	16.90	45	1157	1.29E8	0.00118	32092

Table 3. WS Policy (T=7500)

string	mean wss	max wss	changes of slope	space time product	page fault rate	max interfault time
D**	19.38	51	772	1.18E8	0.000800	80577
Real	19.29	51	742	1.14E8	0.000770	65292

Table 4. WS Policy (T=10000)

string	mean wss	max wss	changes of slope	space time product	page fault rate	max interfault time
D**	21.05	56	613	1.05E8	0.000642	111695
Real	20.90	56	619	1.06E8	0.000648	111695

Table 5. WS Policy (T=15000)

string	mean wss	max wss	changes of slope	space time product	page fault rate	max interfault time
D**	23.71	67	532	1.05E8	0.000570	111695
Real	23.67	67	554	1.07E8	0.000592	111695

When generating artificial strings with two window sizes, it is of interest to investigate the accuracy of the characterization for values of T between the two window sizes used in the generation phase. The results for three intermediate values of T are summarized in Tables 3, and 4, and 5. Not only the first moment results but also the distributions were found to be very close (within 5 percent) to those of the real string [Lee82a].

As shown in the tables, the results for all artificial strings named D** were found to be identical. This observation was generalized in the following theorem, whose proof is lengthy and has therefore been omitted here (it can be found in [Lee82a]).

Theorem

Strings D** generated from two wss characterizations with window sizes T_s and T_l, with $T_s < T_l$, have the same wss characterizations for any T such that $T_s < T < T_l$.

5.3. PFF policy results

That the accuracy of strings S00 and S10 is quite low under the PFF policy is not too surprising. This inaccuracy is due to the fact that, when new pages are taken from the external queue in LIFO order, no further page faults can occur after all pages are brought into memory. However, if in the string generation algorithm we increase the size of the page population at the time a new page is selected, we can adequately reproduce the performance of the real string under the PFF policy. Also double T generation produces accurate artificial strings. This encouraging result is due in part to the similarity between the WS and PFF policies in their dynamic and adaptive allocation of memory to programs. The PFF policy results are summarized in Table 6.

5.4. LRU policy results

The performances of all artificial strings under LRU differ significantly from that of the real string. Even making various modifications to the original simple generation algorithm, accuracies are still unsatisfactory. During the simulation, also the LRU stack distance probability distribution was obtained. The probabilities d_1 of referencing the top of the stack are reported in Table 7. One reason for the inaccuracy of the artificial strings is that the value of the parameter m, the number of page frames allocated to the program in the LRU experiments, was not appropriate. The page fault rate produced by the real program trace with a memory allocation of 21 page frames was more than twice that produced by the WS policy with window size 10000. It is clear that the APL trace does not satisfy all of the assumptions made by Denning and Schwartz [Denn72a]. Assuming that the reference string is stationary is probably unrealistic in this experiment. Similar findings were reported in the literature by other authors (see for example [Smit76a]).

A new value for the parameter m was obtained by trying to match the measured page fault rate of the real string under the WS policy with window size 10,000 references. This new value of m turned out to be 31 page frames. The performance indices obtained from the LRU policy with the new m are given in Table 8. As shown in Table 9, this change resulted in an improvement of almost one order of magnitude in the accuracy of the artificial strings. However, the accuracy is still unsatisfactory. Working-set-oriented artificial strings cannot be reliably used in LRU environments.

Table 6. PFF Policy (I=1543)

string	mean wss	max wss	changes of slope	space time product	page fault rate	max interfault time
S00	55.50	56	56	4.30E7	0.000112	490024
S01	20.97	67	318	1.02E8	0.000648	111695
S02	26.16	64	260	8.83E7	0.000528	111695
S10	55.5	56	56	4.32E7	0.000112	490024
S11	20.97	67	318	1.02E8	0.000648	111695
S12	27.68	64	255	8.89E7	0.000516	111695
Real	20.71	67	417	1.17E8	0.000842	80339
D00	21.91	67	381	1.14E8	0.000776	101010
D01	20.63	67	413	1.18E8	0.000848	80399
D02	20.93	67	409	1.18E8	0.000836	80399
D10	21.91	67	381	1.14E8	0.000776	101010
D11	20.63	67	413	1.18E8	0.000848	80399
D12	20.63	67	412	1.18E8	0.000846	80399

Table 7. LRU Policy (m=21)

string	page fault rate	max interfault time	space time product	d_1 (percent)
S00	0.217	111834	2.28E10	0.0
S01	0.217	111695	2.28E10	0.0
S02	0.217	111695	2.28E10	0.0
S10	0.100	111834	1.05E10	54.4
S11	0.100	111695	1.05E10	54.4
S12	0.099	111695	1.04E10	54.5
Real	0.00146	111416	1.63E08	54.4
D00	0.130	111702	1.36E10	0.0
D01	0.130	111563	1.36E10	0.0
D02	0.130	111563	1.36E10	0.0
D10	0.0605	111702	6.36E09	54.4
D11	0.0606	111563	6.37E09	54.4
D12	0.0601	111563	6.32E09	54.4

Table 8. LRU Policy (m=31)

string	page fault rate	max interfault time	space time product	d_1 (percent)
S00	0.00990	175482	1.55E9	0.0
S01	0.0102	111695	1.59E9	0.0
S02	0.0101	111698	1.58E9	0.0
S10	0.00476	175482	7.54E8	54.4
S11	0.00502	111695	7.94E8	54.4
S12	0.00504	111765	7.96E8	54.5
Real	0.000670	175392	1.19E8	54.4
D00	0.00896	175392	1.40E9	0.0
D01	0.00913	111695	1.43E9	0.0
D02	0.00908	111695	1.42E9	0.0
D10	0.00439	175392	6.96E8	54.4
D11	0.00456	111695	7.23E8	54.4
D12	0.00454	111695	7.20E8	54.4

Table 9. Ratio of LRU Performance Indices

string	page fault rate (artificial/real)		space time product (artificial/real)	
	m=21	m=31	m=21	m=31
S00	148.63	14.78	139.88	13.03
S01	148.63	15.22	139.88	13.36
S02	148.63	15.07	139.88	13.28
S10	68.49	7.10	64.42	6.34
S11	68.49	7.49	64.42	6.67
S12	67.81	7.52	63.80	6.69
D00	89.04	13.37	83.44	11.76
D01	89.04	13.63	83.44	12.02
D02	89.04	13.55	83.44	11.93
D10	41.44	6.55	39.02	5.85
D11	41.51	6.81	39.08	6.08
D12	41.16	6.78	38.77	6.05

6. Conclusions

The tradeoff among the possible choices for a generation algorithm has two aspects : accuracy and complexity. A strategy more sophisticated than the simplest one should definitely be selected if performance indices are unacceptably inaccurate without it. Such a strategy should also be incorporated into the generation algorithm if it adds relatively little to the complexity of the implementation and to the cost of the generation process, while appreciably increasing the accuracies of some of the performance indices of interest.

In a WS environment, since the accuracy of the double T generative model is practically independent of which strategies are chosen, it is clear that simplicity is the primary concern. D00 could be a reasonable candidate; however, D10 has a better coefficient of resilience. Single T generation is not considered here because of the clear advantage of double T generation in the sensitivity of the accuracy to the choice of working set window size.

Double T generation provides acceptable results even in the PFF policy case. It is clear that taking into account the number of distinct pages and using them in FIFO order when a new page is needed is very cost-effective.

The results of the LRU experiments were not very satisfactory, but we should not expect that a WS-based approach for string generation will automatically provide a good accuracy under the LRU policy. Even in this case, double T generation with the minimum number of pages (D10) is the most cost-effective solution among those studied in this paper.

All things considered, a double T generation algorithm is undoubtedly a better choice than any single T generation algorithm. The coefficient of resilience can be taken into account by the algorithm to improve the accuracy in a non-WS environment. At the same time, having the artificial string reference the same number of distinct pages as the real one is also very beneficial. In summary, the D11 strategy should be used.

To improve the accuracy of the model in an LRU environment, incorporating into the algorithm the first order properties of LRU behavior may prove useful. For instance, considering more than one stack distance probability, e.g., d_2 and d_3, in the generation phase of the algorithm may shape the artificial string to be more LRU-like.

The obvious extension of the double T generation approach is a triple T generation algorithm [Ferr82a]. With three reasonably spaced window sizes, the model's accuracy in terms of first moment results as well as of distributions should increase. However, it is not known whether the further gains in model accuracy will justify the increase in the complexity of the generation algorithm.

References

[Chu76a]
W. W. Chu and H. Opderbeck, "Program Behavior and the Page Fault Frequency Replacement Algorithm," Computer 9, 11 (November 1976), 29-38.

[Denn68a]
P. J. Denning, "The Working Set Model of Program Behavior," Comm. ACM, 11, 5 (May 1968), 323-333.

[Denn72a]
P. J. Denning and S. C. Schwartz, "Properties of the Working Set Model," Comm. ACM, 15, 3 (March 1972), 191-198.

[Dutt81a]
C. R. Dutt, "Dynamic Characterization and Reproduction of Program Memory Consumption with Working-Set-Based Generative Model," Master's Report, University of California at Berkeley, August 1981.

[Ferr76a]
D. Ferrari, "The Improvement of Program Behavior," Computer 9, 11 (November 1976), 39-47.

[Ferr81a]
D. Ferrari, "Characterization and Reproduction of the Referencing Dynamics of Programs," Performance '81, F. Kylstra, Ed., North-Holland, Amsterdam, (November 1981), 363-372.

[Ferr81b]
D. Ferrari, "A Generative Model of Working Set Dynamics," Proc. SIGMETRICS Conference on Measurement and Modeling on Computer Systems (September 1981), 52-57.

[Ferr82a]
D. Ferrari, "Working Set Size Strings," CS PROGRES Report No. 82.10, Computer Science Division, University of California at Berkeley, December 1982.

[Lee82a]
T. P. Lee, "Experimental Design of a Generative Model Based on Working Set Size Characterizations," CS PROGRES Report No. 82.6, Computer Science Division, University of California at Berkeley, June 1982.

[Lee82b]
T. P. Lee, "The Properties and Limiting Behavior of Working Set Size Strings and Flatfaults," Report No. UCB/CSD 82/108, Computer Science Division, University of California at Berkeley, November 1982.

[Matt70a]
R. L. Mattson, J. Gescei, D. R., Slutz, and I. L. Traiger, "Evaluation Techniques for Storage Hierarchies," IBM Syst. J., 9, 2 (1970), 79-117.

[Sage73a]
G. R. Sager, "Reproducing Program Memory Reference Behavior," Proc. of Computer Science and Statistics : 7th Annual Symposium on the Interface (October 1973), 41-47.

[Smit76a]
A. J. Smith, "A Modified Working Set Paging Algorithm," IEEE Trans. Computers, C-25, 9 (September 1976), 94-101.

[Spir77a]
J. R. Spirn, "Program Behavior : Models and Measurements," Elsevier, New York, 1977.

SOFTWARE IMPROVEMENT PROGRAM

Opal R. Stroup

Defense Mapping Agency
U.S. Naval Observatory
Washington, DC 20305

This paper summarizes DMA's approach to upgrade its SPERRY Scientific and Technical software while modernizing the Agency's software practices. The objectives of the five-year software improvement program are: increased productivity; improved software quality, maintainability, reliability, and portability; and standardization of software development practices. Both current problems and the program to introduce a modern programming environment, improve existing software and upgrade personnel skills to support the new environment are detailed.

Key words: automated verification; COBOL; DMA; FORTRAN; modern programming; programming standards; software conversion; software improvement; SPERRY 1100; structured programming

1. Introduction

The Defense Mapping Agency (DMA) is currently in the initial stages of a five-year program to upgrade its SPERRY 1100 Scientific & Technical (S&T) software while modernizing the Agency's software production practices. Ultimate objectives of this Software Improvement Program (SIP) are: increased productivity; improved software quality, maintainability, reliability, and portability; and standardization of software development practices. This paper provides an overview of the DMA Software Improvement Program. Before describing the Software Improvement Program, it is appropriate to mention DMA's mission, products, and organization.

2. DMA Mission and Products

DMA's mission is to provide Mapping, Charting and Geodetic (MC&G) support and services for the Secretary of Defense, the Joint Chiefs of Staff and military departments, and other DoD Components through the production and worldwide distribution of maps, charts, precise positioning data, and digital data for strategic and tactical military operations and weapons systems. DMA's mission also includes carrying out statutory responsibilities to provide nautical charts and marine navigational data for use by U.S. vessels and navigators in general.

DMA has five Components including two Production Centers (the Aerospace Center (AC) in St. Louis, MO, and the Hydrographic/Topographic Center (HTC) in Brookmont, Md) which produce the MC&G products and data.

Software may be considered a subproduct of the previously mentioned products rather than an end-product in itself. DMA's software is used to produce, maintain, store, and manipulate data, to drive mapping and charting equipment, produce and validate mathematical models, generate data in digital format, and to perform other functions which create DMA products.

3. Background

The DMA computing environment includes SPERRY 1100 main frames as illustrated in Figure 1[1]. The SPERRY acquisition history from 1972 to the present as well as the systems which will be in place in 1983 are also illustrated. In 1978, DMA initiated the "Phase II Computer Replacement Program" to competitively acquire computing capacity to support the Agency in the 1982-1990 time frame, by replacing the four SPERRY 1100s (one 1108 and one 1100/42 per Center). As part of the replacement activity, DMA and the Federal Conversion Support Center (FCSC) performed a software conversion cost analysis (using the FCSC cost model) for each of several acquisition

Year	AC	HTC
1972	1108	1108
1977	1108 1100/42	1108 1100/42
1980	1100/81 1100/82	1100/81 1100/81
1983	1100/82 2x1 1100/84 4x4 1100/82 2x1 1100/62 2x2	1100/82 2x1 1100/82 1x1 1100/62 MP 2x1 1100/61 1x1

Figure 1. SPERRY ACQUISITION HISTORY

alternatives being considered. To provide the best tradeoff between maximizing competition and avoiding the $30 to $40 million cost of converting DMA's entire applications software inventory to a new target machine, DMA selected the following strategy: upgrade of the SPERRY CPUs, memory, card equipment and printers; competitive acquisition of tapes disks, and terminals, software redesign; competitive contracts for data base conceptual design and implementation; local area networking, and technical support services.

In granting the Delegation of Procurement Authority (DPA) [11][1] GSA suggested that the Agency implement a software improvement program to ensure that DMA will establish an environment which will foster competitive conditions for subsequent procurements. DMA had already recognized a need to improve software and the software development environment and had several on-going related, independent activities in progress. Included in these activities were several Research & Development activities in the area of software development tools.

The Software Improvement Program is intended to consolidate into a single coordinated program many on-going, related activities which have been developing independently. The plan builds on prior Center accomplishments to avoid duplication of effort and to benefit from lessons learned during previous activities [12]. It will initially be implemented for the SPERRY 1100 systems but will later be extended to the minicomputer environment as well.

The operational concept for SPERRY upgrade for the S&T systems calls for transitioning the Agency from a centralized, batch-oriented data processing environment to an interactive processing environment. This includes: acquisition of approximately 500 interactive terminals (HTC - 300, AC - 200); exploitation of data base management concepts and networking; and conversion of non-standard production software to ANS standard languages (ANS FORTRAN X3.9, 1978 [1]; ANS COBOL, X3.23, 1974). Transition from the current batch-oriented environment to interactive environment alone is a significant task. This task is complicated by production software deficiencies, lack of required skills, insufficient automated data processing (ADP) staff, and the absence of a modern programming environment. Several recent independent studies [4,5,6] have identified serious deficiencies in DMA's production software. These deficiencies may be categorized as follows:

1. Multiple versions of production programs.
2. Non-ANS Standard (therefore, nonportable) code.
3. Obsolete coding practices resulting in software which is difficult to maintain.
4. Logical design which is hardware-dependent and inefficient.
5. Poor end-user interface.
6. High error-off rate (much of which results from poor user interface).

Most DMA software developers have received formal university training in disciplines other than computer sciences/software engineering (e.g., physical science, mathematics, earth science, cartography, geography, geodesy, photogrammetry, etc.) and few have training/experience in designing interactive software systems. There is insufficient ADP manpower to perform a massive software redesign while simultaneously supporting normal DMA production. Finally, DMA is in only the initial stages of introducing those tools and techniques which constitute a Modern Programming Environment (MPE). Therefore, we have initiated the SIP and directed it at three areas:

1. Software upgrade. (i.e., software cleanup and software redesign).
2. Upgrading of software development personnel skills.
3. Introduction of an MPE into DMA.

4. Software Upgrade

Improvement of the existing SPERRY 1100 S&T software will require three major tasks:

1. Inventory of existing software
2. Cleanup of selected existing COBOL and FORTRAN software

[1] Figures in brackets indicate the literature references at the end of this paper.

3. Redesign of selected software.

Centers have compiled and continue to refine inventories of existing SPERRY S&T software. Centers are identifying candidates for cleanup and/or redesign using such criteria as: the anticipated life of the software; whether and in what time frame the software is to be off-loaded from the SPERRY 1100; frequency of software use; and criticality of the software to the DMA mission.

The "cleanup" will consist of five major activities: baseline definition, translation, restructuring, validation, and documentation. Center personnel and/or contractors may accomplish cleanup of software. To the extent possible, cleanup is to be accomplished with automated tools rather than manually. The ultimate goal of the cleanup effort is that all SPERRY S&T software will be structured, free of vendor extensions, conform to ANS standards, and documented according to DOD standards.

Baseline definition refers to comprehensive testing of software with retention of results for future comparisons.

Translation is defined as the conversion of existing COBOL or FORTRAN code to ANS COBOL X3.23, 1974, or ANS FORTRAN X3.9, 1978 (FORTRAN 77) respectively, with the removal of vendor extensions. The removal of vendor extensions will in no case result in the loss of function. Once the existing SPERRY 1100 software has been converted to the ANS standard, future conversion to non-SPERRY mainframe or conversion to the ANS subset for minicomputer systems will be more easily accomplished. Translation to ANS Standard will result in more portable, non-vendor-dependent code. DMA does not have a translator tool to accomplish this task. Manual translation would be extremely labor-intensive and error prone. Therefore, DMA plans to require "software redesign/cleanup contractors" to provide such a tool.

Restructuring is the changing of nonstructured code to structured code (i.e., code containing only the following logic structures: Sequence Block, IF-THEN-ELSE, DO UNTIL, DO WHILE, CASE). Restructuring reveals the structure of an algorithm so that its existing code may be maintained, modified, or documented. The restructuring process does not change program logic, and it does not redesign "spaghetti." However, the structured code generated by the structurizer is more readable than the original code and, therefore, is more maintainable.

DMA owns a FORTRAN Automated Verification System (FAVS) [8,9] which provides a FORTRAN structurizer as one of its major subsystems. FAVS is currently being made ready for production use via a maintenance contract with the tool developer, General Research Corporation (GRC). Similarly, a COBOL Automated Verfication System (CAVS) [2] is being developed for DMA. A restructuring capability for COBOL is an option which DMA may exercise.

The validation task refers to the duplication of baseline testing for cleaned up software to ensure that the process did not introduce errors.

The software documentation task consists of both automatic and manual generation of documentation.

During software cleanup, missing or inadequate documentation will be augmented with both automatically and manually generated materials. Automatic generation of reports about code such as static analysis (ANS standard violations, flow analysis errors, portability and flow metrics); complete layout of all files; detailed cross-reference of all statements; and a map of data usage in the COBOL procedure division will be produced as software is cleaned up/redesigned. The purpose of the reports is to assist maintenance programmers in reading and analyzing code and in controlling the impact of program modification.

The FAVS will provide the following documentation for FORTRAN code; invocation summary, common matrices, input/output statements, and cross-reference of external variables for multiple modules; and symbol reports, cross-reference reports, invocation space reports, and invocation bands reports for individual modules [8]. CAVS (when available) will generate the following reports: an indented listing of COBOL source; cross-reference of calling and called programs; cross-reference of program and file interaction; cross-references of program and copy text instruction showning where copy texts are used; cross-reference of program versus linkage section contents; reports showing where all identifiers are defined, set, and used; and a cross-reference of identifiers by record position and programs, showing fields defined, set and used, and when and where identifier names differ [2].

During the inventory phases, Centers identified documentation available for application programs. Missing user documentation is to be written ("manually") during the redesign effort. In cases where contractors perform software redesign, the contractor will prepare such missing documents. In cases where required improvement includes "only" enhancing/preparing documentation, in-house effort will be used.

To allow the Centers to solve a variety of software problems, redesign in broadly defined as any appropriate combination of the following:

1. Rewrite all or portions of the code for interactivity.

2. Redesign of the user interface (leave the code untouched while creating a new "front end" to improve user interface).

3. Optimization of those portions of code which consume the greatest resources (based on results of instrumentation and use of metrics).

4. Rewrite portions of the code to increase reliability.

5. "Scrap" the existing program and rewrite the algorithm using the "structured code" as a basis for understanding.

5. Approach

The DMA approach to the software improvement effort is to contract for the redesign/cleanup of selected software. DMA intends to award Basic Ordering Agreements (BOAs) to all contractors possessing certain corporate experience, personnel experience, and software tool access/experience. A Request for Proposals (RFP) was issued in 1982 for Software Resdesign/cleanup and proposal evaluation is in progress. On a case-by-case basis, DMA will issue delivery orders describing the desiered redesign. A firm fixed-price contract will be awarded for each delivery order on the basis of technical approach and cost. The contractor may be tasked with cleanup only or cleanup plus desired redesign activities. The contractor is being required to use automated tools to translated code to ANS Standard, and restructure it. New code is to be rewritten using only structured programming contructs. In some cases, the contractor will redesign software that has been cleaned-up in-house. In others, unstructured, untranslated code will be the contractor's input -- especially during the early stages of the effort. Since DMA is concurrently attempting to introduce several tools into the sofware development environment, (e.g ., FORTRAN precompiler), contractors will be required to interface with them. For example, a contractor writing structured FORTRAN would be required to use those constructs and delimiters acceptable to the DMATRAN precompiler.

6. MPE Implementation

The improvement (cleanup and redesign) of DMA software will consume a significant amount of resources over a period of five years. To obtain maximum return from this investment, DMA must take actions to ensure that modification/maintenance of the improved software does not result in the introduction of the deficiences discussed above. Moreover, all new software developed must be of the same (or higher) quality as the improved software. Therefore, DMA is concurrently introducing a SPERRY 1100 Modern Programming Environment (MPE). The MPE will include a centralized Production Program Library which will be the repository for production programs and documentation (in human and machine readable form). As each software system is cleaned up and/or redesigned, it will be migrated into the production program library at the appropriate Center and will be placed under control of the newly formed Configuration Control Board (CCB).

Tools to support an MPE implementation plan may be grouped into three general cateogries: (1) conversion aids for software, (2) management aids for existing software, and (3) productivity assistance tools. Three categories of conversion aids are considered in this plan: static analyzers, precompilers, and structuring engines. The FORTRAN Automated Verification System (FAVS) provides each of these tools. FAVS deficiences are currently being corrected so that the tool can be introduced for production use. Similarly, the CAVS will be made available for production use once it is production-ready. A COBOL precompiler which will interface with CAVS is also available. The term precompiler is used here to refer to a tool which simplifies the task of writing structured code in such languages as FORTRAN and COBOL which do not support all of the structured figures.

Two classes of tools to manage existing software (whether developed in-house or by contractors) are being considered by DMA.

The first class of such tools is the code auditor to automatically check for adherence to Center standards [10] for structuring and ANS standards. Use of such a tool would allow DMA to avoid the more labor-intensive, and error-prone manual methods. However, acquisition of a tool is not anticipated prior to introduction of standards.

The second class of tools being considered are configuration control tools which automatically track changes to software and permit only authorized changes to an official version of software. DMA plans to investigate acquisition of such a tool to support the CCB activities. In the interim a manual system is being implemented.

DMA has a great deal of batch-oriented software which requires some form of interactivity. Two approaches can be taken when introducing interactivity. A separate user interface can be written for each program. A second approach is to provide a dialog manager capability to interface with the operating system rather than using COBOL and FORTRAN to do this interface. DMA plans the second approach and has acquired the SPERRY Display Processing System (DPS 1100) which separates the development and use of predefined screens from the application program itself. Center personnel are currently learning to use this tool.

The third major phase of implementing DMA's SPERRY 1100 MPE is the introduction of Standards (which apply to all mainframe and minicomputer software development) into the Centers.

DMA has issued a four-volume set, "DMA Software Life Cycle Standards", which is tutorial in nature, takes into consideration Center differences, conforms to DoD Automated Data Systems (ADS) documentation standards (DoD 7935) and reflects state-of-the-art ADP software practices. It consists of: DMA Software Design and Implementation Standards Manual (SDISM); Structured Programming in FORTRAN [14]; Structured Programming in COBOL [13]; and Structured Walk-Through Guidelines. Volumes II and III detail the simulation of basic structured programming constructs and, thus allow generation of "structured code" without the use of precompilers. Volume IV provides general guidelines for structured walk-throughs for all software life cycle phases. Following Center review, the standards will be introduced in a phased manner. An additional document, "Software Contracting Guidelines" is being developed to assist non-ADP personnel in contracting for software.

7. Skills Upgrade

The third major area addressed by the SIP is the upgrading of the skills of both managers and software developers. Areas of emphasis for managers are: quality assurance, managing structured programming projects, project management and control, state-of-the-art awareness, productivity assurance, and contracting for software. Managers must understand the concepts and methods employed in an MPE since they differ from those of the traditional software projects.

Four areas of training will be emphasized for software developers: (1) SPERRY skills, (2) state-of-the-art awareness, (3) MPE introduction, and (4) new technology. As with management training, a variety of methods (e.g., lectures, seminars, laboratory, video cassette, on-site) will be used. Training topics include: the structured software life cycle, applying standards; use of tools (e.g., FAVS, DMATRAN); designing interactive systems; using terminals; SPERRY refreshers; data base maintenance, query language/report generators; structured life cycle standards; redesigning existing software; and optimization techniques, networking, communications and graphics.

8. Summary

In summary, DMA is commmitted to a five-year Software Improvement Program encompassing three major areas: introduction of a modern programming environment, improvement of existing software, and upgrading development and management skills to support the new environment. General policy includes: adoption/use of structured programming as a standard, maximizing the use of tools to facilitate software development, compliance with ANS COBOL and FORTRAN standards, establishment/use of a centralized program library, adoption/enforcement of software life cycle standards, elimination of multiple versions of common software, establishment of quality assurance groups to ensure adherence to standards, introduction of a Configuration Control Board, introduction of interactivity, and a phased approach to software cleanup/redesign in which high priority software is cleaned up/redesigned first.

Successful implementation will provide the following benefits:

1. A competitive environment in which DMA will not be locked into a single hardware vendor because of the difficulty/costliness of software conversion.

2. A standard software base in which production software is identifiable and maintainable.

3. Standard software development practices within and between Centers.

4. Tools to improve productivity.

5. A modern environment offering increased job satisfaction to software developers.

References

[1] American National Standards Programming Language , FORTRAN, ANS X3.9, 1978.

[2] COBOL Automated Verification System (CAVS): Study Phase , Rome Air Development Center, Griffiss Air Force Base, New York, RADC-TR-81-11, March 1981.

[3] DMAHTC Modern Programming Experience , In-Process Review (IPR) Vu-Graphs, DMAHTC, Brookline, Md, August 1981.

[4] DMA Modern Programming Environment Study , prepared for Rome Air Development Center, Griffiss Air Force Base, New York , RADC-TR-79-343, January 1980.

[5] DMA Programming Support Library (PSL) Pilot Project Interim Evaluation Report , IBM/FSD , prepared for Rome Air Development Center, Griffiss Air Force Base, New York, Contract Number RADC F30602-79-C-0121, November 1980.

[6] DMA Software Conversion Study , HQ DMA, U.S. Naval Observatory, Washington, DC 20315.

[7] DoD Automated Data Systems Documentation Standards , DoD 7935, February 1983.

[8] FORTRAN Automated Verification System User Manual CR-1-754/1, General Research Corporation, prepared for Rome Air Development Center, Griffiss Air Force Base, New York, Contract Number RADC F30602-76-C-0436, April 1980.

[9] FORTRAN Automated Verification System (FAVS) Rome Air Development Center, Griffiss Air Force Base, New York, RADC-TR-78-268, 3 volumes, 1979.

[10] Fife, Dennis, "Computer Software Management: A Primer for Project Management and Quality Control" , NBS Special Publication 500-11, U.S. Department of Commerce, Washington, DC, July 1977.

[11] General Services Administration (GSA) letter , granting DMA a Delegation of Procurement Authority (DPA), 15 July 1981.

[12] Krygiel, A. J., "Lessons Learned on the Road to a Modern Programming Environment" , HQ DMA, U.S. Naval Observatory, Washington, DC 20315.

[13] Structured Programming in COBOL , HQ DMA, U.S. Naval Observatory, Washington, DC 20315, 1982.

[14] Structured Programming in FORTRAN , HQ DMA, U.S. Naval Observatory, Washington, DC 20315, 1982.

SOFTWARE IMPROVEMENT PROGRAM (SIP): A TREATMENT FOR SOFTWARE SENILITY

Carol A. Houtz

Federal Conversion Support Center
Office of Software Development and Information Technology
General Services Administration
5203 Leesburg Pike, Suite 1100
Falls Church, VA 22041

ADP organizations are plagued with high maintenance costs, long delays in responding to users' changing needs, and continued development and maintenance of antiquated, outmoded, and relatively obsolete software. This software can be thought of as being in an advanced state of software senility, a degenerative condition, which if not corrected, will eventually render the software totally useless. A reversal of this situation requires a Software Improvement Program (SIP), which is a treatment for the ills of software senility, and offers a cure for many of the software problems from which most ADP organizations are suffering. A SIP is an incremental and evolutionary approach to modernizing software to maximize its value, quality, efficiency, and effectiveness, while simultaneously preserving the value of past software investments and enabling the organization to capitolize on today's modern ADP technology, as well as future technological advances in the field. This paper describes the SIP philosophy and presents a strategy for implementing a dynamic, ongoing SIP coupled with a sound Software Engineering Technology (SET), to attack the causative factors of the ever-growing software crisis.

Key Words: Software Engineering Technology (SET); software improvement; Software Improvement Program (SIP); software obsolescense; stepwise refinement.

1. Need for Software Improvement

Over the past several decades there have been substantial changes in the automatic data processing (ADP) industry. There have been dramatic increases in hardware productivity, with a significant decrease in the footprint of the hardware configuration due to its reduced size, component modularity, and lowered air-conditioning and electrical consumption. Simultaneously, total ADP costs have continued to rise with the largest costs shifting from hardware to software. This shift in costs is primarily due to substantial automatic data processing equipment (ADPE) price reductions, coupled with increased personnel costs for software development and maintenance activities.

During this same period of high-powered, low-cost, rapidly-advancing ADPE, many ADP organizations are facing a software crisis. Software activities are still labor intensive, with little increases in productivity being realized in software production and maintenance. Resource utilization has shifted from software development activities to maintenance, with over half of all software personnel involved in correcting software errors, modifying software to change its functions or extend its life, and simply keeping the software operational [1][1].

Most existing government software is well over a decade old, with some as much as twenty to twenty-five years old. Much of the software was originally written on second-generation hardware and operating systems, in machine-dependent and nonstandard languages, and have undergone several hardware, operating system, and language conversions. Most of this software was written with little or no utilization of software design, programming, or testing standards, guidelines, or procedures; required substantial operator intervention; utilized sequentially accessed card and tape input and output files; and had minimal, inadequate, or in some cases, a total lack of documentation.

[1] Figures in brackets indicate the literature references at the end of this paper.

Embedded in this aging software were home-grown system utility and operating system features such as sorts, merges, record buffers, copies, and manual restarts. These features were included, of necessity, in the software because most of the features of modern software package utilities and operating systems, which we now take for granted, were not available as packages or in operating systems of that day. Many of these home-grown utilities and operating systems are no longer supported by the developing organization or the vendor, nor is there a readily available and adequate pool of programmers for maintenance of this software.

In the past, bigger or more powerful ADPE configurations, or emulation or simulation has been a quick fix for these software problems. But increasingly, the hardware fix for the software aches and pains has been found to be a fleeting panacea, or a temporary solution at best; and today's modern systems cannot, and do not, support emulation or simulation of the older programming features and practices.

Coupled with these problems of aging software, ADP organizations are plagued with high maintenance costs, long delays in responding to users' changing needs, and continued development and operation of antiquated and underpowered computer software. Productivity increases for ADP organizations with these problems are severely limited, if not impossible to attain, due to the proliferation of archaic software analysis, design, coding, and testing features and techniques; low-level and nonstandard languages; machine or environment dependencies; and custom-written utilities.

This antiquated, aging, outmoded, and relatively obsolete software is in need of modernization. While this software cannot be termed totally obsolete, because it is still operational, it can be thought of as being in an advanced state of software senility. Software senility is a degenerative condition, which if not corrected, will eventually render the software totally useless.

In view of the many and complex, aforementioned software problems, and the emerging trend that the software crisis will continue to grow and worsen, a quick fix or single solution to the problems is not feasible, and a direct conversion from the problem environment to a modern ADPE system and environment is virtually impossible. To solve these problems and combat the software crisis, a program must be instituted to preserve the value of past software investments as much as possible, and provide an incremental and evolutionary approach to modernizing the existing software to maximize its value, quality, effectiveness, and efficiency.

Such a software improvement program (SIP) is described herein as a treatment for the ills of software senility, and offers a cure for many of the software problems from which today's government ADP organizations are suffering. Institutionalization of a sound software engineering technology (SET), coupled with a dynamic, ongoing SIP, can attack the causative factors of the software crisis; and provide the government with viable, modernized, effective, efficient, and high quality ADP systems, capable of capitolizing on today's modern ADP technology, as well as future technological advances in the field.

2. Goals of a SIP

There are many goals for a SIP to achieve. The most important of them being to-

- improve software maintenance and control;
- reduce delays in responding to users' needs;
- improve software quality;
- increase programmer productivity;
- decrease software maintenance costs;
- institutionalize processes;
- change software from a reactive to proactive state;
- extend the software's life; and
- put the organization in a position to take advantage of new and emerging technology.

However, the end goals of a SIP are not only to improve software maintenance and control, but also to achieve as much isolation of function and standardization of interfaces within the software systems as possible. The achievement of these goals is attained through-

- isolating system functions;
- allowing for interchangeability of system functions; and
- facilitating change of elements within function.

Isolating system functions through modularization is a natural step towards avoiding reliance upon one architecture or environment, and increases software maintainability and understandability. As functions are isolated, more design alternatives present themselves and further possibilities of segmentation emerge. Thus, isolation of function holds the key to selecting cost-effective and efficient system alternatives in the future.

Functions should also be interchangeable with alternative design realizations to facilitate functional interfaces. This interchangeability of function, usually achieved through the use of reuseable and standardized modules of code, ensures an easier change in the means of performing a function. For example, exchanging a called module that accesses a tape file for a called module that accesses a disk data set.

Facilitating the change of elements within functions refers to the software's portability and maintainability. Better portability and easier maintainability through the use of single-function, standardized, and reuseable modules of code is paramount to achieving the goals of a SIP. Easier changeability of the software and its functions, increases and ensures more efficient use of the key data processing resources, especially people and machines. Standardizing, modularizing, parameterizing, and documenting the software are several techniques that aid in facilitating the change of elements within functions.

Improving the software's quality (i.e., making the software better), is probably the best available means of achieving the SIP's goals. Software quality is a measure of its excellence, worth, or value against some ideal or standard. Although quality is an ill-defined term, there are many specific properties, or attributes, by which it can be defined or measured [2]. Figure 1 illustrates a proposed hierarchy of the major software quality attributes and their subordinate attributes. Although the subattributes are listed under only one major attribute, it must be stressed that several of them could conceivably be listed under more than one major attribute. For the sake of clarity and to minimize misunderstandings, each subattribute has been listed only once, under the major attribute with which it is most often associated.

It must be noted that it is rarely possible for all software quality attributes or subattributes to be implemented. It is first necessary to define the SIP's goals, and then the improvement objectives for each individual software application. Appropriate tradeoff decisions must then be made among the various quality attributes and subattributes, and the goals and objectives to be achieved. For example, some processing efficiency may have to be sacrificed to achieve more maintainability, and vice versa.

3. Description of a SIP

A SIP can be thought of as a preventive maintenance program for software. Like ADPE preventive maintenance, software must also be periodically and systematically cleaned-up, fine-tuned, optimized, and enhanced to keep it in working order and capable of fulfilling its current and future requirements.

The concept of software improvement is not new, rather it is an outgrowth of normal day-to-day software maintenance projects and an extension of conversion projects. Some types of software improvement, such as realigning code and implementing naming conventions, are presently performed concurrently with such everyday tasks as software modification or maintenance, or for conversion from one computer configuration to another. However, these improvements are traditionally performed on a random, piecemeal basis, without structure or overall software or organizational considerations.

Such improvement decisions are usually made by the individual programmer without the benefit of managerial input. This bottom-up, piecemeal approach to software improvement is unstructured, and usually results in an unsuccessful attempt to cohesively improve the current software and acquire and use modern tools and techniques.

In contrast to the traditional, single-purpose software improvement approach, the modern software improvement approach, as described hereinafter, actually serves multiple purposes. Under the modern software improvement approach, improvements are not performed to meet only a single need or objective, but rather to accomplish several objectives and reconcile multiple problem areas. Also, the decisions as to when software improvement is needed and what types of improvements are needed are not left to the individual programmer or analyst, and the improvements are not performed in a casual manner. Rather, these decisions and the improvement performance are institutionalized as a formal process to which all programmers and analysts must adhere.

While the software improvement concept may not be new, the software improvement approach to building or improving systems, is innovative and more sophisticated than the conventional and more simplistic software life-cycle approach. The software improvement approach to building and improving systems [1] is built on the key assumptions that-

. most major ADP organizations have a decade or more of investment in software;

. most Federal organizations are almost entirely dependent on their software to meet their mission;

. keeping software operational is difficult enough without deviating from that baseline of software to add enhancements or change functions through major redesign or new development, which is thought to be an uncertain and risky business; and

. there is a need to support new applications to keep ADP costs low and service levels high.

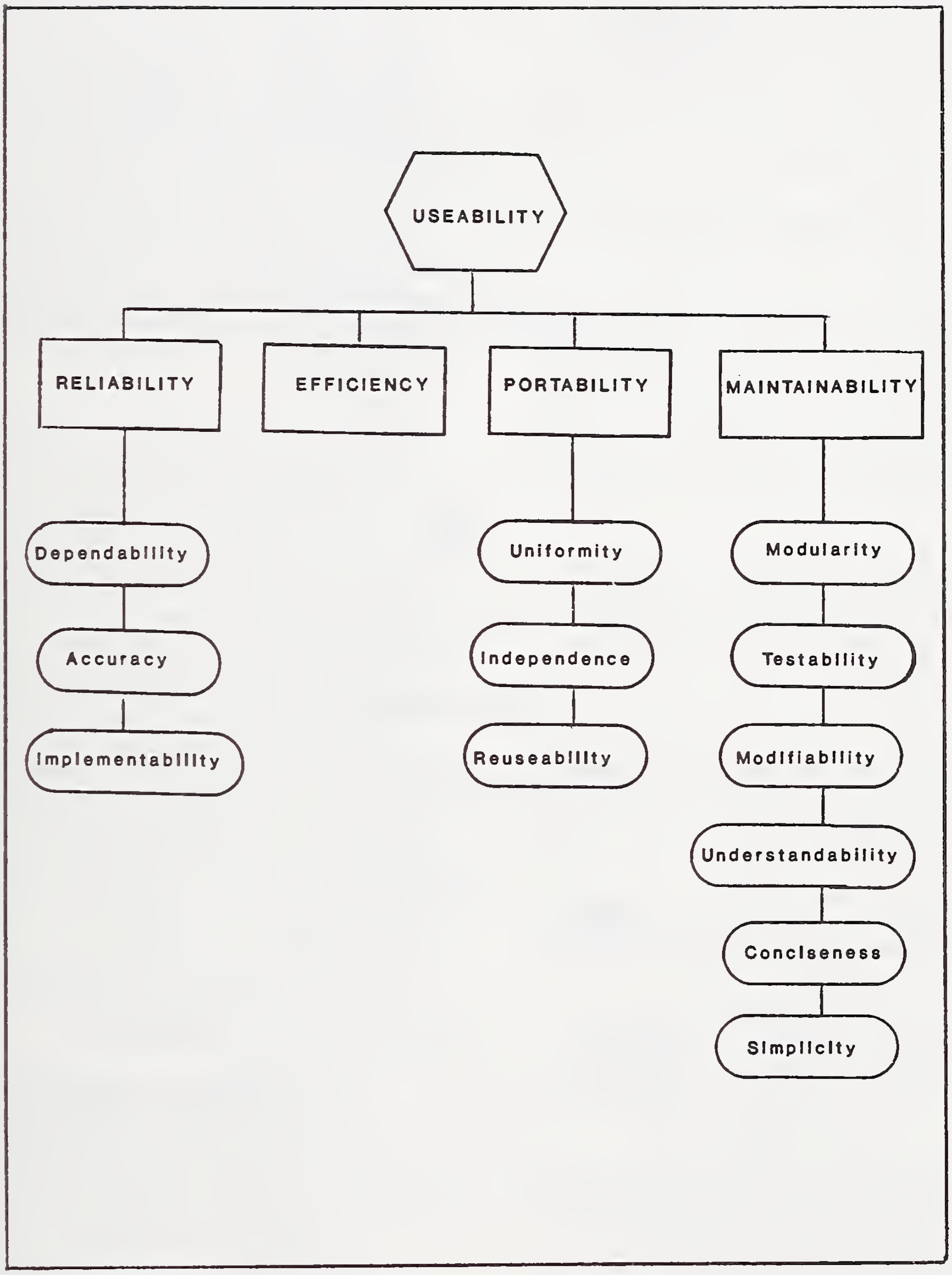


Figure 1. Hierarchy of Software Quality Attributes

The software improvement approach to improving existing systems, or building new systems from existing software, is different from the conventional system development approach in that it recognizes-

- the existence and characteristics of the current systems that support day-to-day operations;
- the existence of other operational systems that may be integrated to replace functions in an existing system;
- the inherent problems in engineering new code in any quantity;
- that existing systems are frequently the only specification of existing processes;
- the need to preserve the testing integrity of the current system while moving to a new or improved system;
- that many faults or deficiencies in an existing system can be accurately and cost-effectively corrected by improving it;
- the need for an orderly, incremental approach to the building or improving of a system that allows for adequate testing, manageable pieces, constant achievement and growth, and progress feedback;
- the need for a useable version of the new or improved system at each stage of development or improvement, allowing for rapid capitalization upon the new system and its components; and
- the virtual impossibility of completely re-engineering or redeveloping very large systems within a reasonable time frame [3].

The universe of software, from which a desired application can be built or improved on, can be conceived as a triangle, as illustrated in Figure 2.

At the apex of the triangle is all of the "software that currently exists" in production today. This software performs the functions that the organization needs to conduct its day-to-day business. Because this software is already working and tested, it has an intrinsic value to the organization and represents the vested interest an organization has in its own software applications. Existing software is usually salvaged, transferred, and incorporated into a new or improved system by purging any undesirable or unnecessary software, leaving some of the software as it is, and improving the remaining software through conversion, refinement, and enhancement activities. While it is typically the easiest and most accurate to test and the least costly to produce, this software is often the most expensive code to maintain because it is usually undocumented and built in a "patchwork" fashion.

At the bottom left-hand corner of the triangle is "other operational software that exists" in other organizations. This external software represents the software packages available from industry or other ADP organizations. While this software may suffer from some deficiencies, it may be modified to fit the organization's needs. Also, many software packages are highly maintainable, well documented, and quite portable, and may not require extensive, if any, modification. External software is usually incorporated into a system by replacing existing code with an existing package. This software is typically somewhere between existing and new software in cost, accuracy, and maintainability, depending on the package's functions and its level of sophistication.

Finally, at the bottom right-hand corner is "new software," which does not yet exist and must therefore be engineered. While this code is typically the easiest to maintain because it is state-of-the-art and newly documented, in terms of accuracy it is normally the most difficult to engineer and the most risky to undertake because there is no existing baseline from which to test or measure. It is also the most costly to produce because it must be engineered from "scratch." This software should be incorporated into a system as a last resort, if transfer of the existing software or replacement with a package is not feasible. Nevertheless, any new software should be engineered using modern programming practices to ensure software that is well documented; fits the application better; is easy to support, read, understand, modify, and enhance; and is less expensive and time consuming to maintain.

The software improvement approach to improving an existing system or constructing a new one, thus is one of-

- determining on a case-by-case basis, the source (e.g., existing internal software, existing external software package, or new software) of the software or software subpiece;
- determining the actions required to modify and/or implement the software (e.g., purge the software from the system; salvage the current software by leaving it alone and moving it as it is; salvage the current software by improving it through conversion, refinement, and/or enhancement; replace the current software with an external software package; or redesign/newly develop the software);

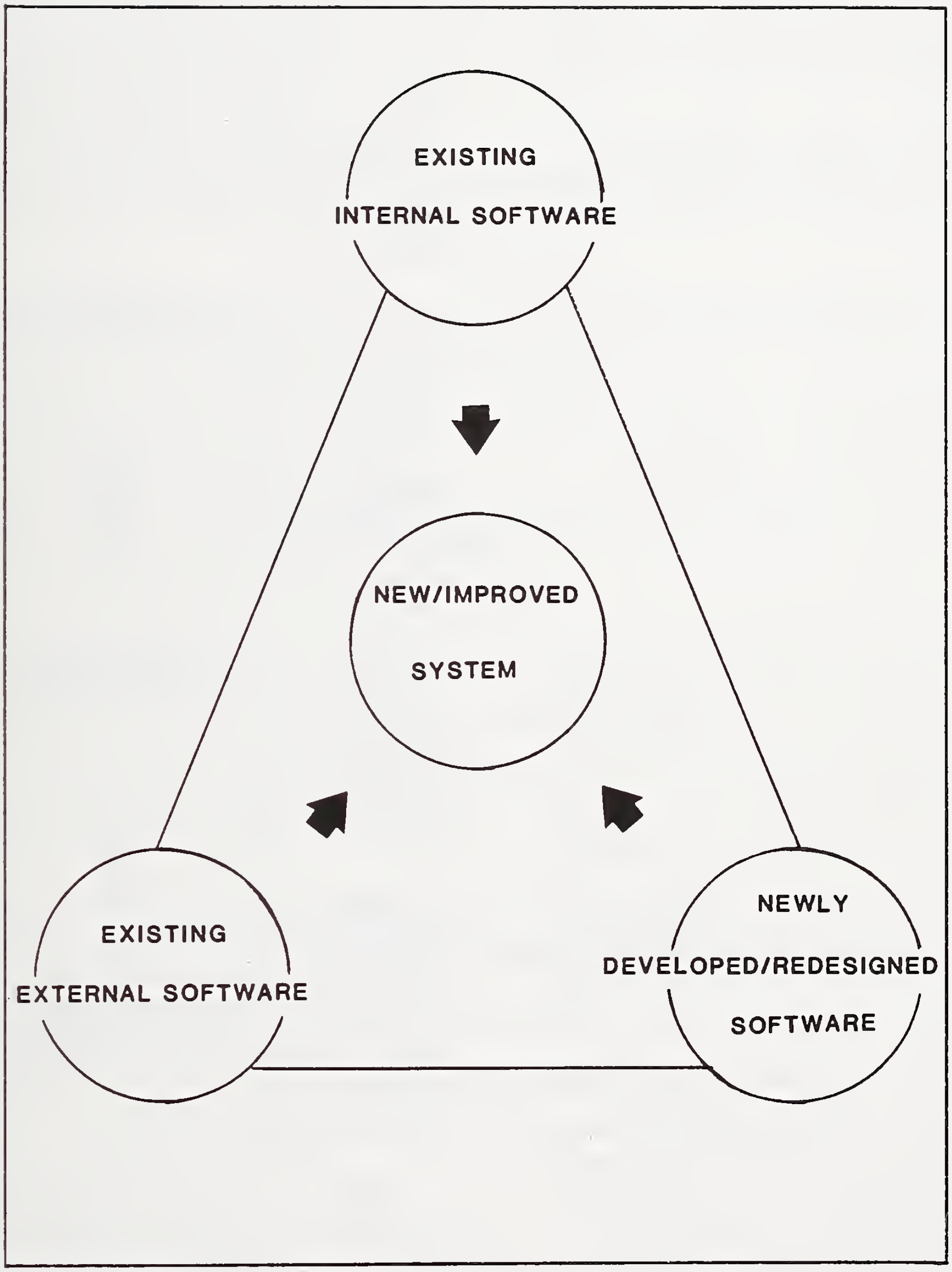


Figure 2. Software Improvement Approach

- assessing the software's source and actions required against the costs, benefits, and risks of each; and, then
- developing a strategy or plan for the software's purge, transfer, improvement, integration, and/or redesign/new development into the improved or newly constructed system.

In summary, four basic advantages of the software improvement approach to improving or building a system over the conventional system development approach are that it-

- minimizes uncertainty and risk by maximizing the utilization of testable components;
- preserves the value of past software investment as much as possible and avoids the dangers of failure inherent in the "tear-it-down-and-start-anew" approach;
- enables the project to be broken into small, manageable pieces with an operational system at each phase; and
- is iterative in nature, which allows for the tasks performed to be repeated in an orderly, incremental fashion with constant achievement, growth, and feedback, until the overall objectives of the project are met.

4. Stepwise Refinement

Because of the large amount of software that exists in most ADP organizations, most software can't be improved in one "lump sum." Thus, the software must be divided into smaller, more manageable increments, that progress through the software improvement process as a work unit. Increments can be based on system, subsystem, or project boundaries, or by functional areas (e.g., input, edit, file update, report generation, or error handling). The key is to subdivide the software minimizing the interfaces between the groups. The absense, or minimization, of increment interfaces makes the improvements for each increment more independent, and allows the concurrent improvement of several increments at a time.

Also, because of the vast differences that may exist between the current and desired software environments, most needed improvements can't be accomplished in one "quantum leap." Thus, the improvements for each increment are accomplished in multiple steps or releases [3] of logically related sets of improvement activities that are performed at one time. Improvements are normally subdivided by activity type into the three basic releases of conversion, refinement, and enhancement as depicted in Figure 3.

As illustrated in Figure 3, the software improvement activities under these three releases range from simple translation of code to complete re-engineering of existing systems. The software improvement activities flow from one release to the next; thus, there is no "clear cut" dividing line between each release, and some functional overlap is inevitable.

The decisions as to the number of releases necessary to improve each increment, and the improvement activities to be performed in each release, are dependent on the size of the increment, overall number and type of improvements required, and priority of the improvements. Improvement activities can be combined into one large release, or further subdivided into multiple mini-releases as illustrated in Figure 4. Figure 4 also illustrates the stepwise refinement approach to upgrading and modernizing the software, with continual advancement and the opportunity between each stp to reevaluate the SIP plans and results, and to introduce changes as necessary.

Software improvement conversion activities transform the software, without functional change, standardizing it and making it environment independent. Without standardization and independence, the next two releases, refinement and enhancement, would be extremely difficult, if not impossible, to accomplish. Standardized software, that is as independent as possible, lends itself to manipulation by automated means and proceduralized processes, and facilitates flexibility for future requirements (e.g., moving to a new environment).

Software improvement refinement activities modernize the software to a state-of-the-art status and improve software maintainability and programmer productivity. Refinement is a prerequisite for software enhancement to ensure enhancements are not being made to unmaintainable software with obsolete coding features (e.g., EXAMINE or ALTER statements in COBOL), or outdated or incorrect functional requirements.

Software improvement enhancement activities optimize the value, quality, efficiency, and effectiveness of the software enabling easier technical redesigns, easier addition of modern "technological" features and capabilities, and more efficient and effective use of resources. Without enhancement, the standardized and modernized software may still not function efficiently or effectively, or fulfill the user's desired requirements.

Improvement activities do not have to be subdivided into these three basic releases or follow the suggested release flow. They can be combined into one large release, or further subdivided into multiple mini-releases. The decisions as to the number of releases necessary to improve each increment, and the improvement activities to be performed in each release, are dependent on the size of the increment, overall

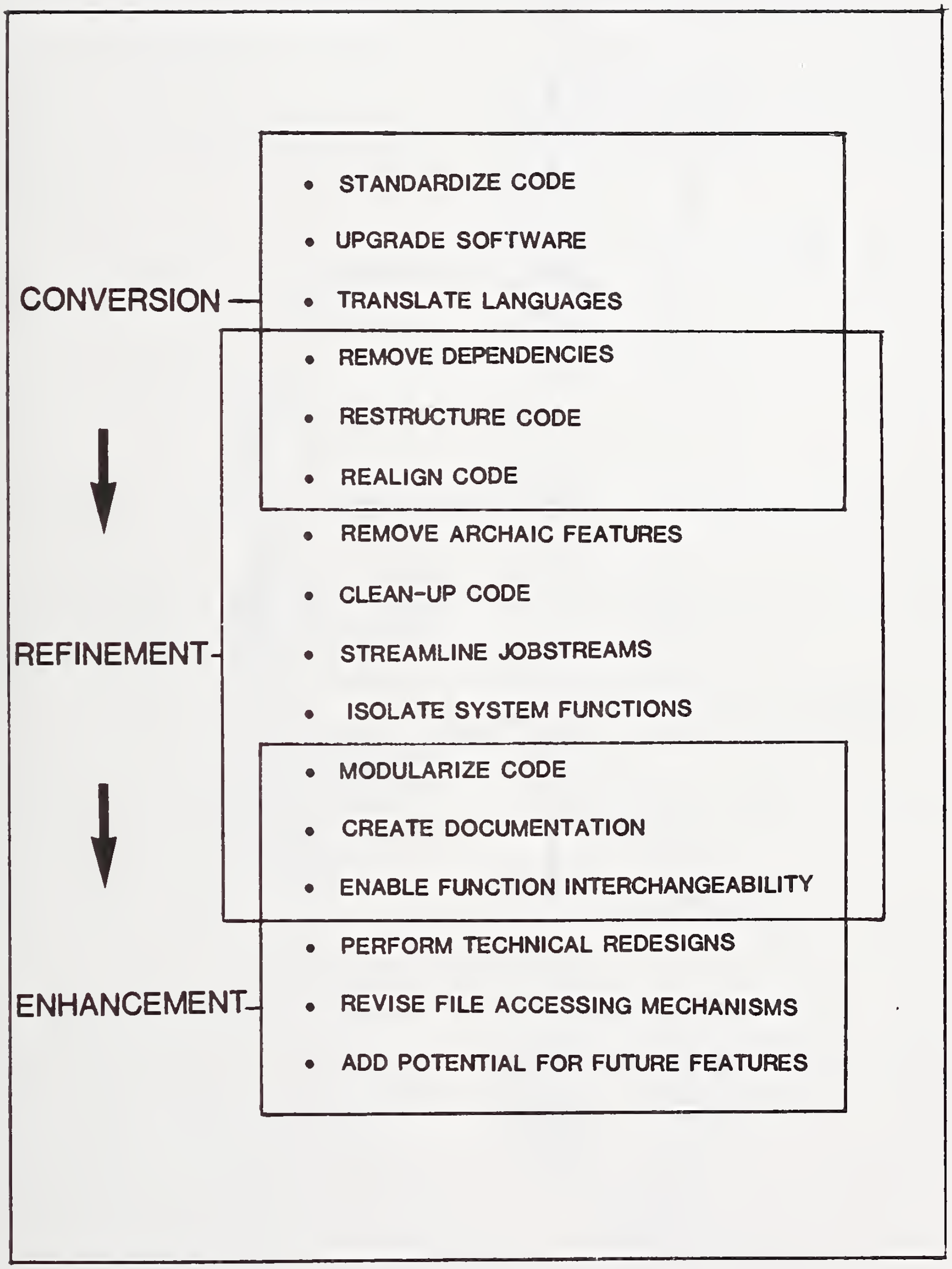


Figure 3. Typical Software Improvement Release Flow and Activities

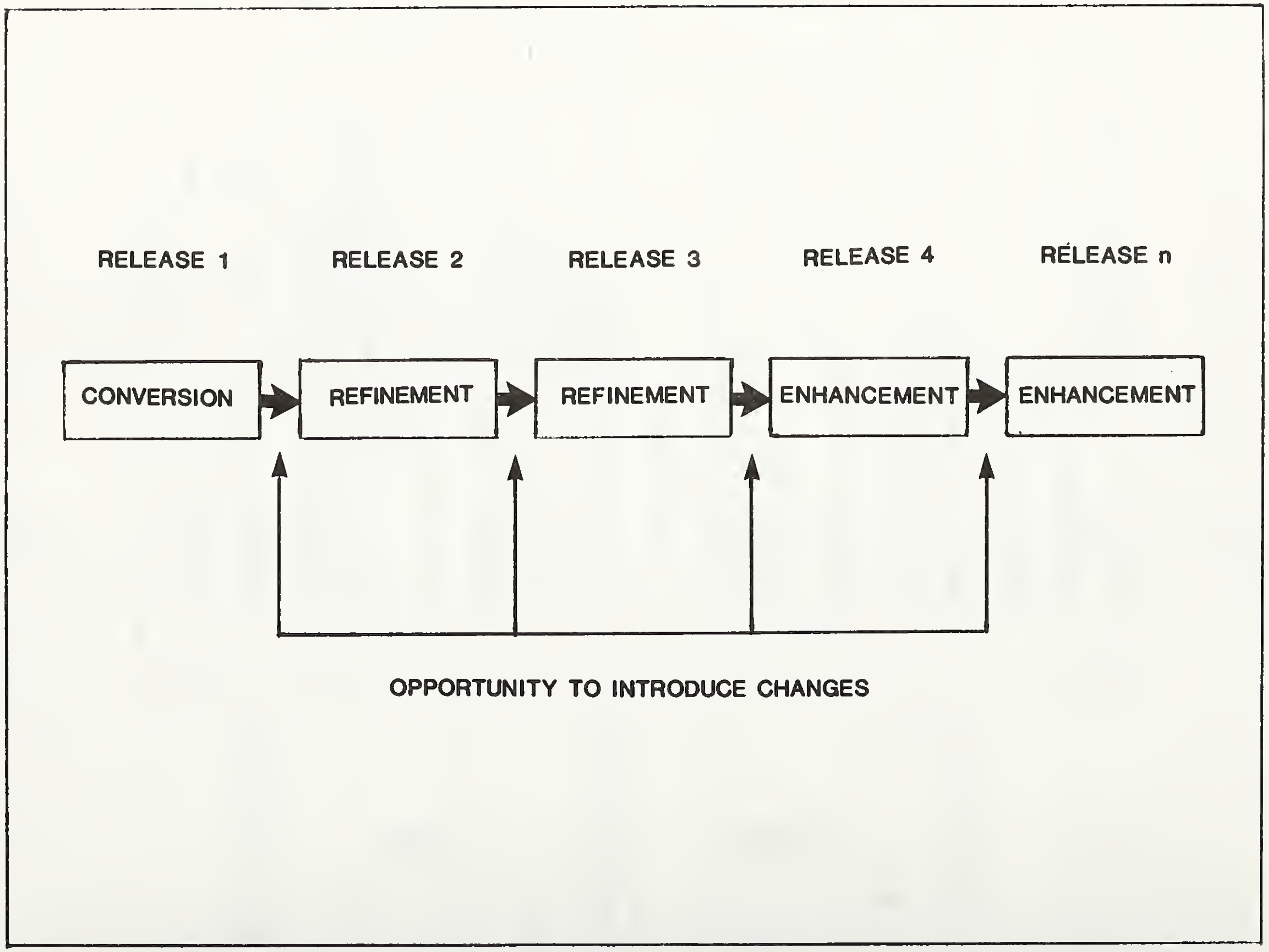


Figure 4. Typical Incremental Release Flow

number and type of improvements required, and priority of the improvements.

5. Major Benefits of a SIP

The tasks required during the software improvement process encompass a broad range of activities, and at a minimum include-

- performing a software inventory and analysis;
- developing SIP plan(s);
- establishing engineering elements;
- preparing work packages;
- preparing test data sets;
- developing software improvement release specifications;
- improving the software;
- unit and system testing the improved software;
- documenting the software;
- acceptance testing the improved software; and
- transitioning the improved software into production.

From this list of tasks, it is clear that a SIP is labor, management, machine-resource, and, possibly, deadline intensive. However, in spite of the problems that will inevitably arise, a SIP can be successfully engineered and prove highly beneficial to the organization.

The advantage of utilizing state-of-the-art technological advances, such as teleprocessing, data base management systems (DBMS), and mass storage, is one such benefit. Also, a SIP provides the capability to use this modern technology without being "locked in" to architectural or environmental dependencies.

Another benefit is the potential for more efficient and effective programmer productivity. Existing software, after improvement, can be maintained much more efficiently and the programmer's span of control should be greatly increased. That is, after software improvement, a programmer can maintain significantly more lines of code or system functions due to the increased maintainability and understandability of the improved software. The result is increased availability of an organization's most scarce resource -- skilled programmers. A SIP more efficiently uses key resources, both people and machines. More readily available junior personnel can be used for both new development and maintenance, with improved productivity, lower risk, and less training. The more senior personnel can be used for more advanced tasks such as systems design or analysis, or tool evaluation and selection.

Additionally, the incorporation of a Software Engineering Technology (SET), consisting of a synchronized set of software standards and guidelines, procedures, tools, quality assurance, and training implemented through and coupled with a dynamic, ongoing SIP, simplifies the learning required of programmers and analysts. The simplified learning enables the institutionalization of a single training program for a common methodology and consolidated goals and objectives.

More efficient use of the ADPE is also possible because the state-of-the-art ADPE will not simply emulate obsolete or out-of-date functions. Rather, it will perform the technologically advanced activities for which it was designed.

Many additional benefits can be achieved with a thorough, comprehensive, and well-planned, -analyzed, and -managed SIP. Some of these include-

- improved user service levels;
- more flexibility for future requirements;
- the capability for automatic documentation and/or code generation;
- enhanced error recovery, system debugging, testing, data integrity, and security features;
- increased software quality (i.e., reliability, efficiency, portability, and/or maintainability);
- improved quality of software end-products (e.g., reports, statistics, and programs); and
- a synchronized, formalized, and tested SET for the SIP.

6. SIP and SET Interrelationship

As previously discussed, the SIP works closely, and in tandem with a SET. A SET, as depicted in Figure 5, consists of a synchronized group of five equally important software engineering elements:

- standards and guidelines;
- procedures;
- tools;
- quality assurance (QA); and
- training.

These five software engineering elements direct and control all software activities throughout the software's life cycle [4], and

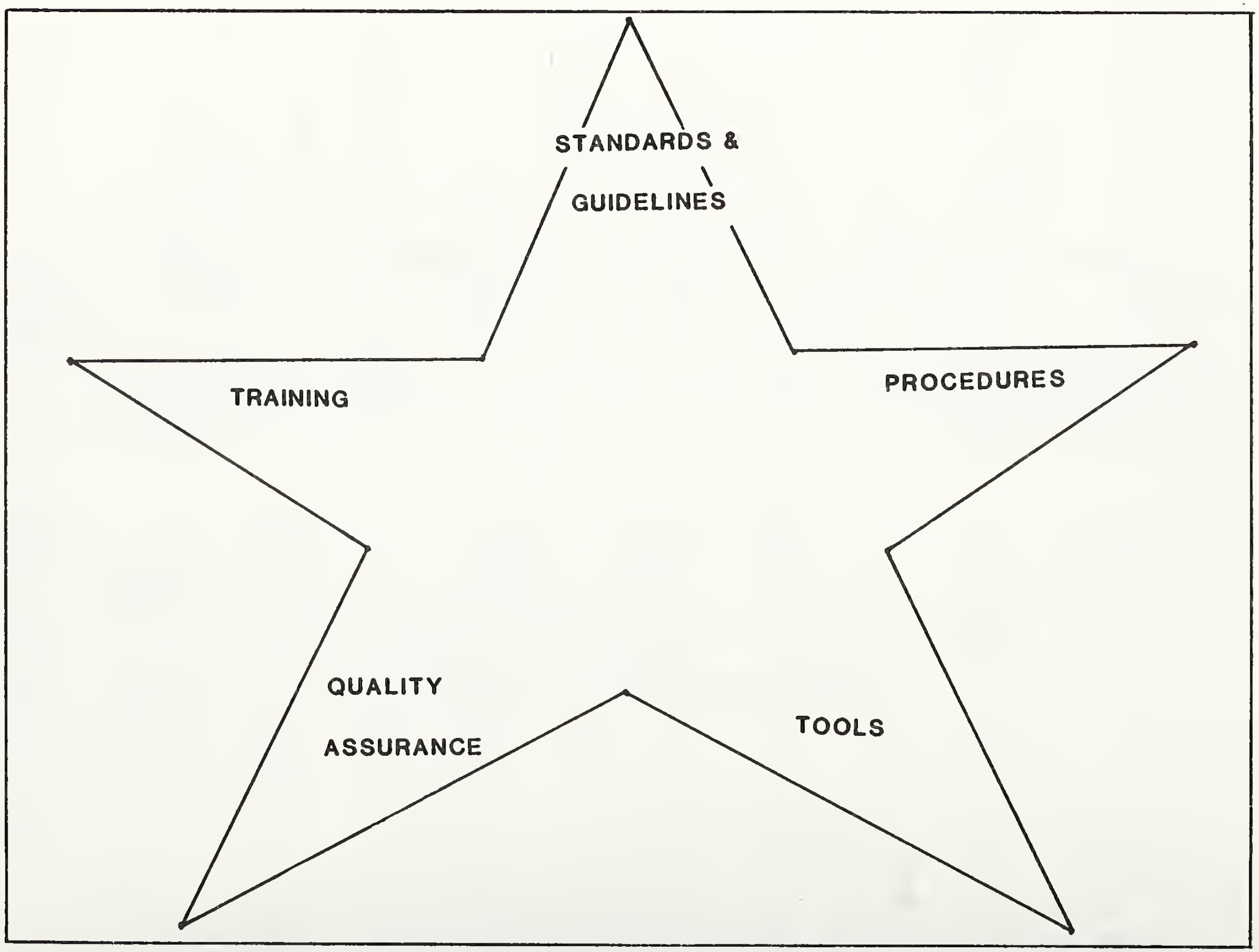


Figure 5. Software Engineering Technology Engineering Elements

for different software engineering or re-engineering purposes (e.g., software development, maintenance, improvement, conversion, or redesign). Thus, the SET addresses, on an organization-wide basis, the software engineering methods, metrics, and latest controls for managing an installation's software activities, while the SIP addresses the upgrading of the existing software (i.e., programs, modules, job streams, files, and documentation) with regard to the SET baseline.

The same five engineering elements of a SET are required, in a specialized sense, for a SIP. The establishment of these five engineering elements as a formalized SET is paramount to the successful implementation of a SIP. Improvement and installation standards, guidelines, and procedures are required to standardize the software activities and the software, so they can be measured, controlled, and improved. Specialized improvement tools are both necessary and desirable to increase programmer productivity, enforce systemization, and improve controls. QA is required to quantitatively prove that the SIP is a viable and worthwhile effort, ensure that the improvements made actually resolve the problems, identify and measure the resultant improvements and benefits, and control and enforce the quality of the software and the improvement performance. Finally, training and retraining are also necessary, for without them successful accomplishment of the SIP would be next to impossible and the improved software and methodologies would quickly degrade.

The establishment of a SIP and a SET are separate, but interrelated and coordinated processes. Each can be established independently of the other, but each has a controlling or influencing effect on the other. That is, the standards and guidelines, procedures, tools, QA, and training established for the installation as a SET, define the framework for, and provide a baseline from which, the SIP can operate. In this sense, the SIP cannot, for example, set standards that oppose those instituted in the SET, or use tools that conflict with the tool's technology chosen for the organization and established in the SET.

Conversely, standards and guidelines, procedures, tools, QA, and training, when established in a SIP, limit the choices of the SET. For example, the SET cannot institute software or installation standards different from those just implemented by a SIP, or maintained and enforced by the improvement tools. If either the SIP or SET institute engineering elements without considering the organizational impact, or short- and long-term consequences, the resulting software and engineering activities will, at best, be chaotic and consist of a "patchwork" of styles, structure, and standards.

A SIP and a SET should be established together, one complementing the other. This double-barreled approach to resolving software problems can be thought of as a Software and Technology Modernization Program (STMP). Figure 6 illustrates a typical interrelationship of the SIP and SET as integral parts of a STMP. Implementing a STMP thus becomes an effort to-

- identify needed program-unique or organization-wide engineering elements (i.e., standards and guidelines, procedures, tools, QA mechanisms, and training);
- consider long-term software activities and consequences (e.g., ADPE and software compatibility, the upward compatibility of software changes, and technological advancements), and analyze the full spectrum of impact (e.g., across engineering activities, or between projects), before adopting any specific standards, procedures, etc.;
- isolate the program-unique engineering elements for the SIP from the organization-wide engineering elements for the SET;
- adopt and institute as part of the SIP, the program-unique engineering elements; and
- adopt and institute as part of the SET, the organization-wide engineering elements.

7. Synosis of a SIP

A synopsis of the six key principles of a SIP are:

- Evolutionary Growth: That is, build on your past software investment as much as possible by purging some of the software, leaving some of it alone, replacing some software with packages, improving most of the software, and redesigning or newly developing only that which is absolutely necessary.
- Incremental Improvement: Minimize the risk of failure and make the SIP more manageable by grouping the software, through functional decomposition, into smaller, logical subpieces with minimal interfaces.
- Top-Down Planning with Bottom-Up Input: Plan in a hierarchical manner, from a general, overall SIP level, progressing to more specific, increment levels. Allow for continual feedback, analysis, and evaluation of improvement plans, results, and methodologies.
- SIP Pilot Project: Prototype the improvements, engineering elements, and methodologies on a small scale to empirically demonstrate the feasibility

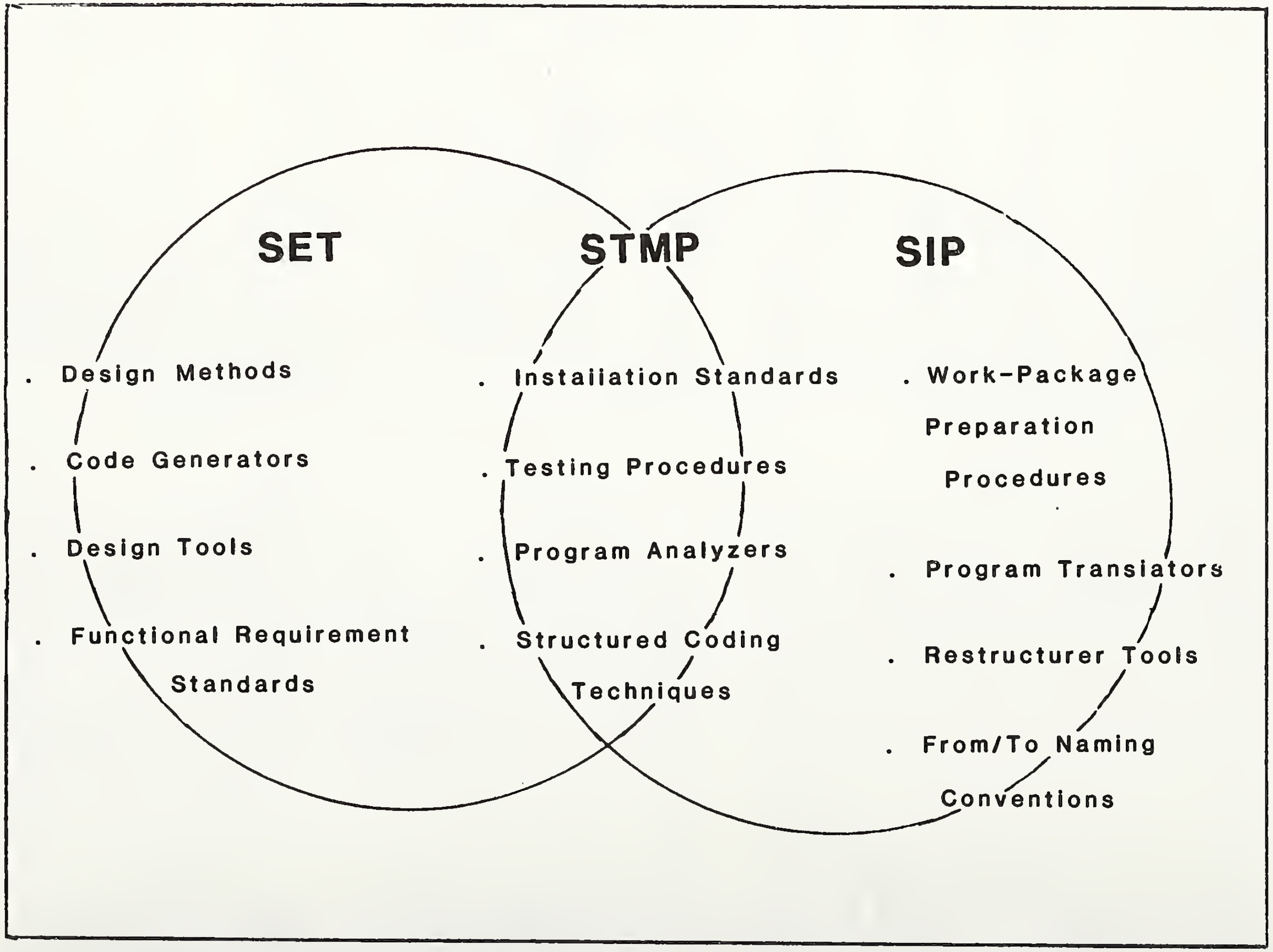


Figure 6. Example of Typical SET and SIP Interrelationship under a STMP

and success of the improvements, methodologies, and plans; and to help solve, early in the SIP life cycle, any technical problems that may occur.

- Release Specifications: Subdivide the improvements required for each increment into logically related sets of improvement activities that can be performed at one time (e.g., conversion, refinement, and enhancement). Develop release specifications which direct and control the improvements to be performed for each release of an increment. Be sure to include in the specifications the specific improvements required for each individual system, subsystem, program, module, job stream, and/or file in the increment, as well as the required deliverables, standards of performance, and acceptance criteria.

- Engineering Elements (SET): Establish within the SIP, or as a baseline SET, formalized engineering elements (i.e., standards and guidelines, procedures, tools, QA, and training) to be implemented, employed, and enforced by the SIP.

While the SIP guidelines presented here may not seem to be "earthshaking," they are a less risky, more formalized means of modernizing the organization's information processing, and have been found to be "tried-and-true." The use of these SIP guidelines is strongly encouraged, and in concert with a strong SET, will promote more uniform, thorough, cost-effective, and efficient software and software engineering activities.

8. SIP Case Studies

The most successful organizations have recognized that their software is a key asset, which must be developed, managed, controlled, and maintained with as much care and attention as their other important assets. That is why these organizations have invested, or are investing, significant resources into SIP's, using principles similar to those described here.

The experiences of these organizations should not be considered unique. The concept of a SIP is indeed a valid alternative to the two traditional choices of "don't-touch-it-or-it-will-fall-apart" and "tear-it-down-and-start-anew." Unless the functions are changing, substantial redesigns may not be necessary, and a SIP may solve immediate, as well as long-range ADP problems. It is an alternative with principles and procedures applicable to most government agencies, and must be given serious consideration.

Several organizations have successfully established and undertaken SIP's. Some examples of these organizations are the Office of Personnel Management (OPM) in Washington, DC; Raytheon Service Company in Boston, MA; NCR Corporation in San Diego County, CA; and the San Diego County Department of Education in San Diego, CA.

Several years ago, OPM undertook a SIP with gratifying results. Many of its ADP systems were developed in Assembler language for an RCA Spectra 70/45, and when converted to COBOL, still reflected the second generation logic of the earlier Assembler code. OPM decided to adopt a controlled system improvement approach, migrating in steps from a second to third generation system. The decision was also made to convert the Assembler code to ANSI COBOL to simplify maintenance and enhance portability considerations [5].

Similarly, in the late seventies and early eighties, Raytheon undertook a SIP with the primary objective of developing, implementing, and perfecting a reuseable code methodology for accelerating applications development. By using reuseable code, 40 to 60 percent of the redundancy in their business applications development was eliminated, and maintenance was substantially improved [6].

In late 1976, the NCR Corporation undertook a large scale Quality Improvement Program (QIP) for a major set of systems software for over 103 separate products. This software set included operating systems, compilers, peripheral software, data utilities, and telecommunications handlers, and totaled over 1.3 million lines of source code. The QIP was initiated to provide improvements in the software base and to take advantage of recent advances in the state-of-the-art of software engineering. NCR found several major favorable effects resulting from the QIP, such as a substantial reduction in outstanding problems in the software base, a reduction in the average number of error reports per month, total elimination of problem backlogs, and a significant reduction in late responses to problem reports. All of these improvements are reflected in an improved perception of the quality of the software, and allowed NCR to make a very substantial redirection in funds from support of existing products to the development of new ones [2].

Finally, the San Diego County Department of Education's ADP data center cut its maintenance time by 70 percent as a result of a SIP. This startling reduction in the data center's program-maintenance load has resulted primarily from the decision to adopt and convert to structured design and programming techniques, with ongoing and formalized ADP training in these same areas. While the primary emphasis in this effort was on redesigning and replacing the existing systems, rather than salvaging the existing systems, the six key principles of a SIP were basically adhered to [7].

Besides the organizations that have established and undertaken successful SIP's, several Federal organizations are currently in the initial steps of establishing SIP's. Several of these organizations include the Social Security Administration (SSA) in Baltimore, MD; Veterans Administration's (VA) Data Processing Center (DPC) in Austin, TX; and Defense Mapping Agency (DMA) in Washington, DC.

The SIP being established by the SSA is a key example where the two traditional alternatives are infeasible. SSA is in the process of transitioning its more than 16,689 computer programs, containing over 11 million lines of code, from a "survival" mode to a state-of-the-art environment. To leave the systems alone will only result in further ADP system deterioration and seriously jeopardize the agency's ability to perform its basic mission. Conversely, to redesign all software would be extremely risky, require maximum reinvestment of resources, and require more time than SSA has to survive the current crisis. Thus, the key is an incremental, evolutionary improvement approach, aimed at a recovery of SSA's heavy investment in its software, and the ability to take advantage of new ADP technological advances [8].

The VA's Austin DPC is another example where top management has recognized that the efficiency and effectiveness of their mission is a function of their software. The Austin DPC has over three million lines of code of application software. Like software in most government agencies and industry, this software was not developed overnight; rather it has evolved over many years, with layer upon layer of modification making it even more complex, unmangeable, and unmaintainable. Together with a hardware-upgrade and SET initiative, the Austin DPC is initiating a SIP to cut escalating ADP costs, improve the quality of service to the veteran, and better support far-reaching management decisions [9].

The DMA is currently in the initial stages of a five-year program to upgrade its software and modernize it software production practices. The ultimate objectives of this SIP are to increase productivity, improve software quality, and standardize software development practices. DMA's SIP encompasses the three major areas of introducing of a modern programming environment, improving existing software, and upgrading development and management skills to support the new environment [10].

Several more organizations establishing SIP's are Tupperware in Orlando, FL; Ford Aerospace and Communications Corporation in Sunnyvale, CA; and New Jersey State Government.

From the preceeding discussion, it should be clear that organizations who can no longer afford outdated and inefficient information processing, want to combat the software crisis, want to stop "software "senility" in its tracks, and, on the whole, need to modernize their software and software engineering technologies must establish a SIP. A commitment to undertake a SIP begins with top management, progresses through the ADP organization, and, ultimately, ends with the user. Top management commitment to the SIP is a major factor for its success, and manifests itself in three forms:

. First, top management must acknowledge that a software problem really exists, and resolve to correct it.

. Second, top management must be willing to "put their money where their mouth is." That is, they must not offer only "lip service," but be willing to devote the resources necessary to implement the SIP. Resources include people, dollars, time, ADPE, tools, and other miscellaneous supplies and materials.

. Third, top management must actively support the SIP and ADP organization by helping to advertise the SIP goals, objectives, benefits, and achievements, and by gaining user involvement and support.

There are three documents currently published on the subject of software improvement that may be of further interest to organizations contemplating a SIP. These documents are listed in the references [1, 11, 12] and contain more detailed information on the need for software improvement, planning and implementing a SIP, and the actual software improvement process.

References

[1] Office of Software Development, General Services Administration, Software Improvement - A Needed Process in the Federal Government, Report No. OSD-81-102, June 3, 1981.

[2] Woodmancy, Donald A., "A Software Quality Improvement Program," NCR Corporation, San Diego, CA, IEEE Catalog No. 79CH1479-5/C, 1979.

[3] Office of Software Development, General Services Administration, Long Range Plan, Report No. OSD-82-104, April 9, 1982.

[4] Federal Software Testing Center, General Services Administration, Establishing a Software Engineering Technology (SET), Report No. OSD/FSTC-83/014, June 1983.

[5] Cooper, Roger, "Upgrading Federal Computers Through Existing Systems," Government Executive, August 1979.

[6] Lanergan, Robert G. and Poynton, Brian A., "Reusable Code: The Application Development Technique of the Future," Raytheon Service Company.

[7] Beeler, Jeffry, "Manager Cuts Maintenance Time by 70%," Computerworld, January 31, 1983.

[8] Social Security Administration, U.S. Departmetn of Health and Human Services, Systems Modernization Plan, February 1982.

[9] Austin, Texas Data Processing Center, Veterans Administration, Software Improvement Program (SIP) Macroplan, August 1983.

[10] Stroup, Opal R., DMA Software Improvement Program, Talking Paper for Federal DP Expo (Session E-2) "Dealing with Obsolescence: Conversion and Upgrading," DMA, U.S. Naval Observatory, Washington DC, April 12, 1983.

[11] Federal Conversion Support Center, General Services Administration, Guidelines for Planning and Implementing a Software Improvement Program (SIP), Report No. OSD/FCSC-83/004, May 1983.

[12] Federal Conversion Support Center, General Services Administration, The Software Improvement (SI) Process --Its Phases and Tasks, Report No. OSD/FCSC-83/006, July 1983.

SOFTWARE IMPROVEMENT
THROUGH
AUTOMATED NORMALIZATION

Dr. Michael G. Walker

WBG, Inc.
1489 Chain Bridge Road
McLean, Virginia 22101

1. Introduction

Many major ADP centers have a decade or more invested in their applications software and the organizations they support are almost totally dependent upon its operation. The software "works" in that its logic is basically correct and it supports the organization's mission. In most centers, merely keeping the software operational is such a difficult task that any change to the baseline, either to enhance or add functions, is reasonably perceived as both risky and expensive. However, the software must be improved to keep ADP costs low and service reliable. Software improvement is the process of modernizing software by retaining its fundamental logic while upgrading its reliability, economy, and flexibility. The goal of improvement is to posture software to take advantage of new technology.

The software improvement approach differs from traditional system development because software improvement recognizes the following factors:

0 the fact that current systems "work";
0 the substantial investment in current systems;
0 the technical and informational properties of current systems;
0 the interchangeability of some of these properties with functions of other systems;
0 the risks in developing new programs in any but trivial quantities;
0 the only specifications for many existing systems are the systems themselves;
0 the desirability for an orderly improvement process;
0 the need for a usable system at every stage of improvement, and
0 the cost and technical difficulties of re-engineering a large system within a reasonable time frame.

The combination and interaction of the above factors suggest that most organizations with substantial investments in software should improve their software incrementally and not redevelop the total system. Some software is so old and so patched that it can't be improved. However, where software improvement is possible it has the following advantages:

1. minimizes risk by retaining an operable baseline;

2. preserves value of past software investment;

3. upgrades in small, manageable, and testable elements; and

4. progresses by iteratively enhancing the existing baseline.

Organizations with large software investments, like the Federal Government, are underwriting ambitious programs of software improvement. For instance, the Social Security Administration, the Defense Mapping Agency and Veterans Administration have all initiated large software improvement programs to modernize their computer software including its features, capabilities, and engineering practices. Also the General Services Administration Office of Software Development has initiated a government-wide software improvement program to upgrade software systems across the Federal Government. (Software Improvement - a Needed Process in the Federal Government GSA Report No. OSD-81-102.)

Normalization is the initial and most fundamental technical process in achieving software improvement. It is the process of standardizing existing systems thus making them maintainable, enhanceable and portable as well as posturing them to take advantage of new technology. After a system is normalized, it can be more easily refined and optimized, new functions can be added in a predictable manner, and economies can be realized.

This paper defines normalization by: (1) demonstrating how it improves a typical applications capability (our example will show a COBOL application that utilizes both a Teleprocessing Monitor(TP) and a Data Base Management System (DBMS)), (2) describing how it can be automated, and (3) providing two examples in which it has been used to improve software.

2. Normalization: A Definition

Normalization is the process of improving software by making it:

0 enhanceable

0 maintainable; and

0 portable.

Enhanceable software can be economically improved either by adding new functions or by refining and optimizing current functions. Maintainable software can be quickly and reliably fixed when it breaks or can be modified when changes are requested. Portable software can be moved from one computing environment to another to take advantage of the technological economies and advancements of new hardware or software. Normalization improves these attributes of existing software.

Exhibit 1 outlines a typical COBOL system before it has been improved through normalization. The applications software is intertwined with the supporting environment, as well as knotted with extensions to a vendor's COBOL, with interfaces to a DBMS and to TP.

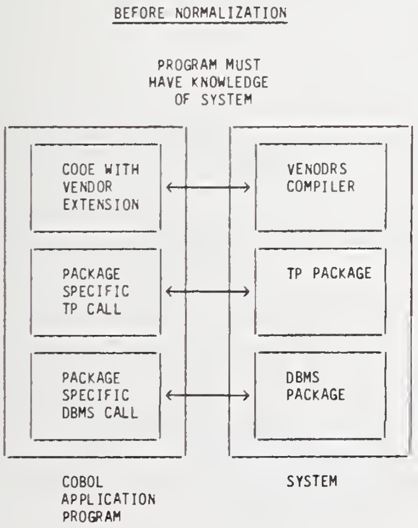

BEFORE NORMALIZATION
PROGRAM MUST HAVE KNOWLEDGE OF SYSTEM
COOE WITH VENDOR EXTENSION
VENODRS COMPILER
PACKAGE SPECIFIC TP CALL
TP PACKAGE
PACKAGE SPECIFIC DBMS CALL
DBMS PACKAGE
COBOL APPLICATION PROGRAM
SYSTEM

Exhibit 1. Typical COBOL On-Line System Before Normalization

The applications software must "know" the characteristics of the DBMS, the TP, and the operations control language (OCL) peculiar to the computing environment. The program syntax has extensions that frequently are inconsistent with ANS standards. Program semantics use data in ways peculiar to the source vendor's architecture. Program logic is linked to the DBMS, the TP and the operating systems by imbedding "knowledge" of these packages throughout the code. As a result, any significant change in either the code or the packages has a profound impact upon the other. The problems of maintaining and enhancing such a system are significant.

Exhibit 2 depicts a normalized capability in which the knots have been untied and the applications "freed" from the host system through standardization and information "hiding." The program syntax has been normalized into ANS approved constructs and the semantics normalized so the programs will produce the same results if they are executed on other vendors computers. The knowledge of the DBMS, the TP, and the operating system has been separated from the application logic and embedded in bridge programs. This means the application programs can be changed without impacting the system, and that the DBMS or TP can be changed without altering the program's logic. The interface knowledge is "hidden" in the bridge programs. There is some processing overhead involved in hiding the information. The cost benefits come from reductions in software costs. Normalization trades improvement in software quality for extra computer cycles.

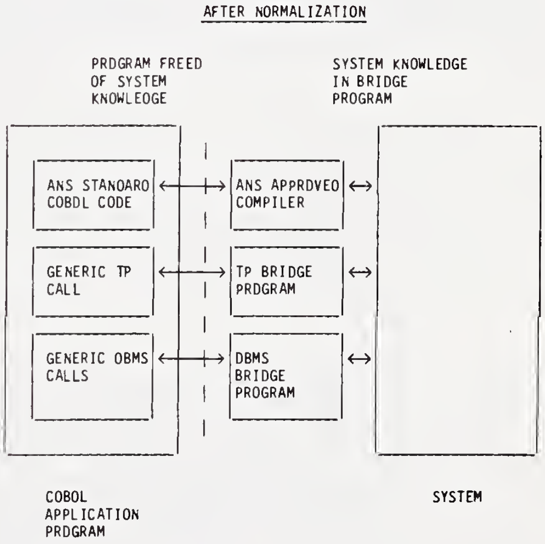


Exhibit 2. Typical COBOL On-Line Capability After Normalization

Exhibit 3 outlines a total picture of how normalization improves applications software.

The system has been made more enhanceable, maintainable, and portable as a result of normalization. The software has been improved while preserving the legacy and investment in the applications logic. This improvement was managed without redevelopment. Thus, the risks of developing new code were avoided; the costs and technical difficulties of engineering a new system were obviated; and the investment in the current system was preserved. The following sections will describe how to normalize a system and will offer two examples of a successful normalization.

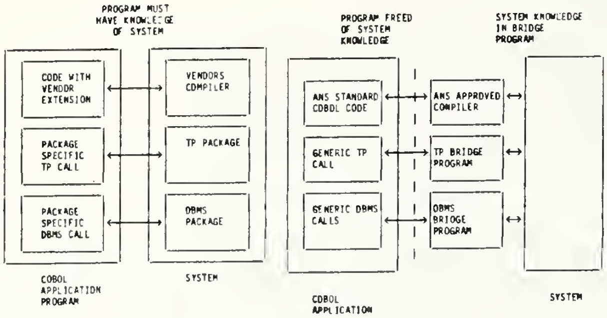


Exhibit 3. Comparison of a System Before and After Normalizing

3. Normalization: The Process

To improve a software applications system, each system element should be normalized. This may include upgrading the programs, DBMS, TP Monitor, Operating System and documentation as shown in Exhibit 4. The basic program logic will not be changed; the software foundation will. The programs must be normalized both in terms of their syntax and semantics. This entails translating non-ANS statements into ANS and normalizing the use of data. The DBMS must be normalized in terms of program calls and interfaces, data schema, and data representation. This entails standardizing the interface between the program's logic and the system's handling of data. The TP monitor must be normalized in terms of program calls and interfaces, message handling capabilities, and screen formats. This entails standardizing the interface between the program's logic and the system's handling of transactions. The operating control language must be normalized in terms of standardized utilities and job streams. This entails standardizing the interface between the program's logic and the operating system. The program documentation must be normalized in terms of applying a standardized and automated documentation tool that will keep it accurate and current.

PROGRAMS

0 Syntax
0 Semantics

OATA BASE MANAGEMENT SYSTEM

0 Program Calls
0 Oata Schema
0 Oata

TELEPROCESSING MONITOR

0 Program Calls
0 Message Handling
0 Screen Formats

OPERATING CONTROL LANGUAGE

0 Utilities
0 OCL

OOCUMENTATION

0 Automated Program Oocumentation

Exhibit 4. Software Elements That Must Be Normalized

To address each system element, three tasks must be completed. These tasks, outlined in Exhibit 5, are: Evaluation, Translation, and Verification and Validation. Each must be completed using each system element shown in Exhibit 4.

EVALUATION

0 identify each change to each element

0 summarize and analyze these changes

TRANSLATION

0 make each needed change

0 incorporate change into system

VERIFY ANO VALIOATE

0 test translated system

0 certify functional equivalence

Exhibit 5. Tasks That Must Be Accomplished to Normalize a System.

Evaluation (EVAL) identifies the individual changes to each software element and the manner in which each change must be made. EVAL also summarizes these changes and performs an analysis on the summary. This analysis provides a quantitative picture of the difficulty of normalization and provides statistics on which to base the schedule and cost of the subsequent tasks. Exhibit 6 shows how an automated evaluation technology parses the system elements, identifies each change, and summarizes the changes.

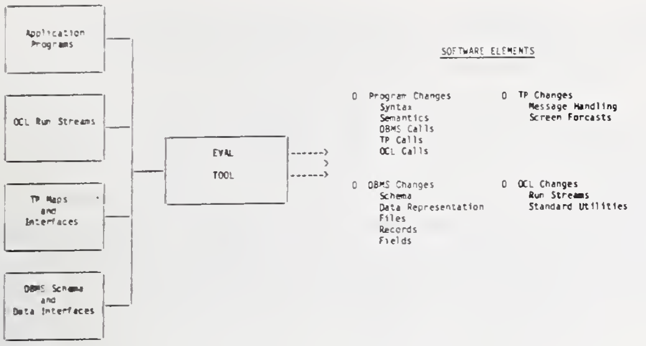


Exhibit 6. Automated Software Evaluation

Translation(TRAN) makes each change identified in the evaluation and incorporates these changes into the system. The translator(s) must change the programs, DBMS, TP, OCL, and documentation. Thus there is a specific TRAN tool for each of these elements. TRAN changes the programs into ANS COBOL and normalizes their use of data; builds DBMS bridge programs, normalizes DDL schema, and translates data into the new models and representations; builds TP screen formats and standardized interfaces; builds new OCL streams; and generates new documentation.

Validation and Verification(V&V) tests each change and validates that the normalized system is the functional equivalent of the pre-normalized system. V&V testing includes both unit level and system level testing and verifies the normalized software at the unit level and the system level. V&V provides a testbed for testing, verifying, and validating the normalization.

Exhibit 7 outlines a five-by-three array that represents the work that must be accomplished to normalize a system. The intersection of each system element and each normalization task represents work that must be accomplished to complete the normalization. Its composite represents the total effort to do the job. Additionally, each individual piece of work must be coordinated into a broader, integrated effort. If they are not, the sum of the costs of the parts may be much greater than the value returned by the total effort. When this happens normalization, like system development, becomes a risky expensive undertaking.

	NORMALIZATION TASKS		
SOFTWARE ELEMENTS	EVALUATION	TRANSLATION	VERIFICATION & VALIDATION
APPLICATIONS			
DATA BASE			
TRANSACTION PROCESSING			
OPERATORS CONTROL			
DOCUMENTATION			

Exhibit 7. Relationships of Elements That Must Be Normalized to Task Needed to Normalize.

Exhibit 8 compares three methods of organizing a normalization, the manual approach, the partial automated approach, and the integrated automated approach. These approaches differ in the amount of technology each uses. The cost and risk of normalization can be significantly reduced as the technology for normalizing is improved from disjointed manual processes to integrated automated processes.

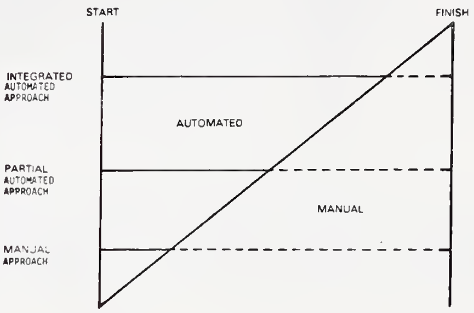


Exhibit 8. Comparison of Three Methods of Organizing a Normalization Project

The most risky approach is the conventional manual process that relies totally on manual effort to complete the job. This method features all of the problems associated with finding, keeping, and managing large staffs plus the problems of managing the quality of unpredictable, error-prone, manual techniques. The manual approach entails the risks of redevelopment without offering the potential benefits of a "new" system.

The partial automated approach is an improvement over the manual. It relies on selecting tools from various sources and combining these tools with man-hours needed to use them. This method is an improvement over the manual because it applies some modern technology (e.g., it substitutes predictable technology for manually efforts). However, it is plagued with the management problems of coordinating disjointed tools and techniques, adjusting uneven staffing requirements (some tools are more labor saving than others), and integrating these variables into a total project effort. Often the gains of using disjointed tools and techniques are lost to the inefficiencies of managing and coordinating them.

The preferable methodology is the integrated automated approach depicted in Exhibit 8 and which features tools that not only fit each specific work area but are also part of a total technical process. This preferable technology is tool or technique intensive and "fills" each box with a large amount of automation and a much smaller amount of manual effort. The tools reinforce each other in an integrated and supportive manner. For example, the evaluation tools (EVAL) provide information and a framework that the translation tools (TRAN) use to make the changes identified in the evaluation phase. These changes provide the audit trail that the verification and validation tools (V&V) use to verify the quality of the normalized system.

Exhibits 9a and 9b show the work flow for an automated normalization. It starts with the planning step and ends with a fully tested system. Each step in the process is assisted by techniques whose products intergrate with subsequent steps. The steps should be aided by tools as follows:

PHASE	STEP	TOOL
Planning	Planning	EVAL
	Baselining	VAV
	Build Conversion Unit Packages	Program Support Library (PSL)
Conversion	Translate Programs, DBMS, TP	TRAN (PGM, DBMS, TP)
	Convert Test Files	TRAN (DBMS)
	Convert Test JCL	TRAN (OCL)
	Unit Test	V&V
	Convert System Files	TRAN (DBMS)
	Convert System OCL	TRAN (OCL)
	System Test	VAV
Implementation	Acceptance Test	VAV
	Document System	DOC

Exhibit 9a. Phases, Steps & Tools of Normalization Work Flow and Work Processes

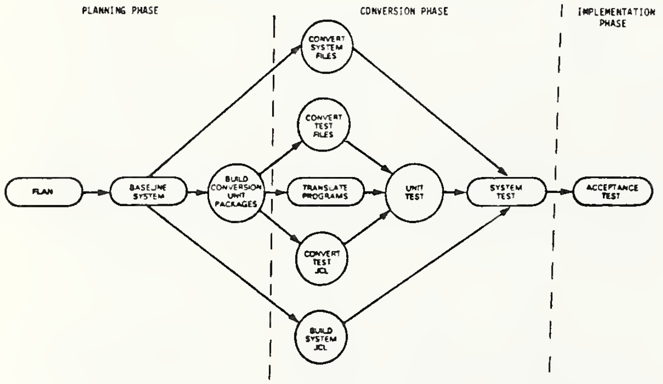


Exhibit 9b. Normalization Work Flow and Work Processes

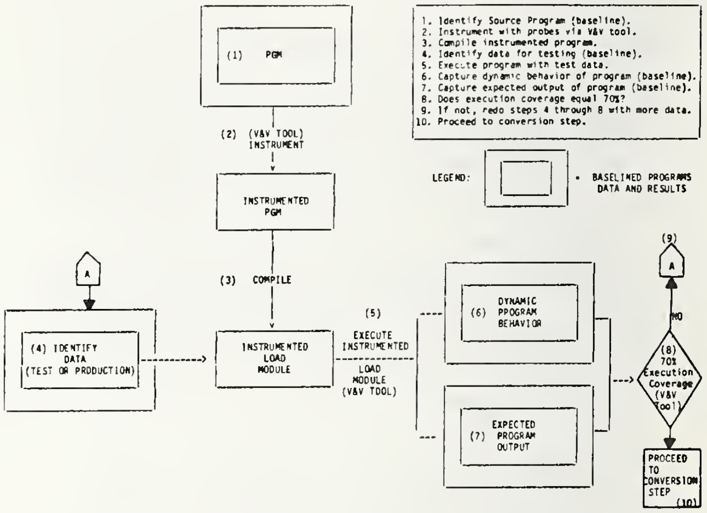
The amount of effort for the total job and the distribution of work per phase depends upon the size of the job and the difficulty of the normalization. However, the integrated approach makes the effort predictable and the risk manageable.

Two Key Steps

The two key steps in managing a successful normalization are baselining and systems testing. Normalization is not simply performing the translation step. Although automated translation is necessary it is certainly not sufficient to manage the normalization process. Every step should be automated but the baselining and testing steps are especially critical because these steps can have the most deleterious impact on the normalization effect.

Baselining and System Testing can be very labor intensive and very unpredictable if they are not aided by automation. As much as fifty percent of an improvement effort can be spent in baselining and system testing; and if these steps are not managed correctly, the normalization effort will fail. Automation makes these steps both manageable and predictable.

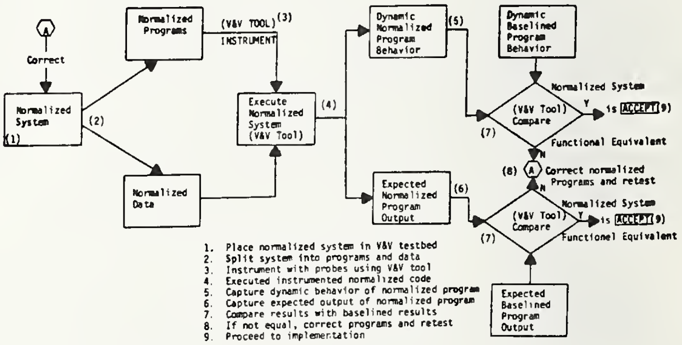
The baselining step, shown in Exhibit 10, prepares the software for normalization. It produces a definitive specification, an empirically derived picture of the current production software system. It includes not only each of the software elements but also the dynamic behavior of these elements as they are tested, including test data, test transactions and processing results.


1. Identify Source Program (baseline).
2. Instrument with probes via V&V tool.
3. Compile instrumented program.
4. Identify data for testing (baseline).
5. Execute program with test data.
6. Capture dynamic behavior of program (baseline).
7. Capture expected output of program (baseline).
8. Does execution coverage equal 70%?
9. If not, redo steps 4 through 8 with more data.
10. Proceed to conversion step.

(1) PGM
(2) (V&V TOOL) INSTRUMENT
INSTRUMENTED PGM
(3) COMPILE
A
(4) IDENTIFY DATA (TEST OR PRODUCTION)
INSTRUMENTED LOAD MODULE
(5) EXECUTE INSTRUMENTED LOAD MODULE (V&V TOOL)
(6) DYNAMIC PROGRAM BEHAVIOR
(7) EXPECTED PROGRAM OUTPUT
(8) 70% Execution Coverage (V&V Tool)
NO
(9) A
PROCEED TO CONVERSION STEP (10)
LEGEND:
• BASELINED PROGRAMS DATA AND RESULTS

Exhibit 10. Baselining Step in Planning Process

The System Test, shown in Exhibit 11, verifies that the normalized system is functionally equivalent to the baselined system. It uses the definitive specification provided by the baseline to verify that the improved normalized system produces "exactly" the same answers as the baselined copy.


1. Place normalized system in V&V testbed
2. Split system into programs and data
3. Instrument with probes using V&V tool
4. Executed instrumented normalized code
5. Capture dynamic behavior of normalized program
6. Capture expected output of normalized program
7. Compare results with baselined results
8. If not equal, correct programs and retest
9. Proceed to implementation

A
Correct
Normalized System (1)
(2)
Normalized Programs
(V&V TOOL) (3) INSTRUMENT
Normalized Data
Execute Normalized System (V&V Tool)
(4)
Dynamic Normalized Program Behavior
(5)
Expected Normalized Program Output
(6)
Dynamic Baselined Program Behavior
Expected Baselined Program Output
(V&V Tool) Compare
(7)
Y
N
Normalized System is ACCEPT (9) Functional Equivalent
(8) A Correct normalized Programs and retest
Normalized System is ACCEPT (9) Functionel Equivalent

Exhibit 11. System Test Step in Conversion Phase

One of the great assets of normalization is this ability to produce a definitive specification and to test against it. After a baseline is captured, it is the specification. The improved system must produce the exact answers as the baseline. When it does, the system is accepted as functionally equivalent. Because the V&V technology helps automate this process, normalization verification can be predictably planned and managed and its cost controlled.

The automated V&V technology is used in both baselining and testing. For Baselining, the V&V technology is used: (1) to instrument the software with probes; (2) to run test data against the instrumented programs; and (3) to capture the execution behavior, the flow of the programs, and the expected output. Exhibit 10 shows a step-by-step flow of the baselining process. The system elements, test data, execution stream, and expected output are all captured and are all part of the baseline. This totality forms the acceptance criteria for the normalized system. It is used as the definitive specification in the System Test step to verify that the normalized system is the functional equivalent of the baselined system.

For System Testing, the V&V technology is used as shown in Exhibit 11. The normalized software is instrumented and tesed under the same conditions as the baselined software. The system behavior during the test is captured and stored. This behavior is compared with the results produced in the baselining step. Any discrepancies in the results of the System Tests and the Baseline Test are identified and the code is made functionally equivalent.

Each phase and step of the normalization process can be automated and controlled to produce predictable results. Through automated normalization, software can be improved in a manageable fashion and the risk and expense of improving software can be tightly managed and controlled.

The following section provides two examples of software improvement projects that used automated normalization to modernize software. The first example involves using normalization to minimize the risk of transition in the hardware acquisition of a DoD Agency. The second example involves using normalization to move a major online capability from one computing environment to another.

4. Normalization as Part of a Hardware Selection Strategy

EXAMPLE 1. A DoD Agency is replacing 35 large mainframes that are part of a nationally distributed capability and that are interconnected by an internal telecommunications network and are connected to a DOD wide digital data network. This hardware inventory includes IBM, and HONEYWELL computers and the software inventory includes approximately eight million lines of COBOL and assembler, several data base management systems, and several TP monitors. The Agency is planning to replace most of these computers in a single buy and to convert to a single vendor's DBMS and TP software package.

The risks of converting both the hardware and software in one step was so great the Agency wisely decided to minimize the software related risks by improving software prior to its conversion. The Agency chose to normalize its inventory of programs into ANS 74 COBOL and a single DBMS, and operating systems, and TP monitor prior to acquisition. It used normalization to help in both hardware procurement and software improvement. In its hardqare selection, it used software and normalization to create a multi-target benchmark package that is executable across many vendor lines and to provide this package for the Live Test Demonstration (LTD) of each vendor's proposed hardware solutions.

In its software improvement, the agency used normalization to upgrade its most critical programs into a more portable, maintainable, and enhanceable status prior to hardware implementation. Thus, by normalizing, the Agency met its two basic goals:

1. Minimizing transition risk by putting its most critical software into a normalized posture prior to hardware installation, helping ensure the Agency of a smoother risk free transition period.

2. Maximizing hardware competition by providing every vendor with the same normalized easily executable LTD benchmark package that will minimize the bidding vendors conversion costs and maximize their ability to compete for government business. This competition will reduce the Agency's life cycle costs and increase its systems quality by multiplying its options of possible vendors.

Exhibit 12 shows how the Agency's benchmark package for the LTD was selected, normalized and verified.

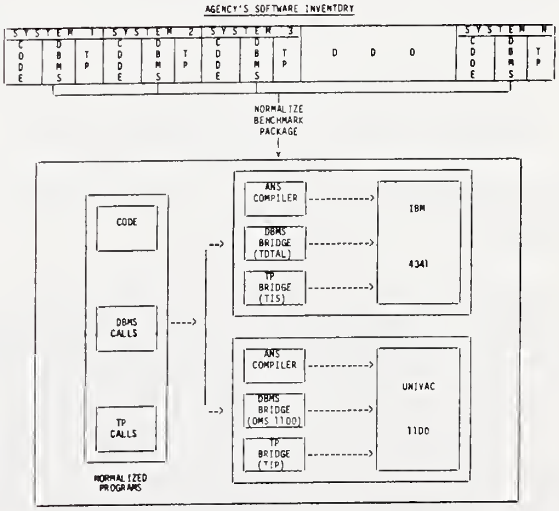


Exhibit 12. Normalizing + Verifying Benchmark Package for LTD

0 Representative programs and transactions were selected from across the Agency's inventory of systems. These programs represent the types of processing the Agency will conduct over the next ten years and the types of processing the hardware must support both in terms of the functions, volume, and mix of transactions.

0 Since the Agency's processing demands are impacted by sudden and extreme surges in transaction volumes, the LTD must also model these heavy but sporadic transaction loads. The programs selected for the benchmark package were normalized into a multi-target status. This normalization was verified on IBM in a CINCOM TOTAL MVS environment and in UNIVAC in a DMS 1100 environment. These are two radically different environments. This process involved normalizing:

  0 languages;
  0 interfaces;
  0 DBMS;
  0 TP;
  0 OS;
  0 hardware; and
  0 documentation.

  Automation makes the process technically possible.

0 Since the normalizing process is automated, the verification took place in a very timely manner matching the writing and release of the hardware RFP. The benchmark package that was provided vendors for the LTD included the normalized programs and data together with detailed specifications for the bridges.

The multi-target normalization established a common baseline of programs and transactions that vendors benchmarked in the LTD thus providing the Agency with two valuable measures:

1. The performance of the normalized software and

2. A comparison of the performance of vendor's hardware using the normalized software.

Because the code is normalized, it can also be used to provide performance measurement after installation of the new equipment.

Normalization helped meet the goals of maximum competition and minimum conversion risk. Competition is maximized by using portable software for the LTD thus minimizing vendors conversion costs and risks. This should induce more vendors to compete to supply the Agency with hardware. Life cycle costs are minimized by having a unified software environment normalized for ease of maintenance and enhancement and fitted to the hardware so processing loads can be met. This should reduce the software maintenance and enhancement costs that contribute approximately 80% of the Agency's ADP operational expenses.

Normalization is a Migration Strategy

EXAMPLE 2. A U.S. Government civilian Agency operates a nationally distributed network of terminals connected to a large data center. As the initial step in its computing modernization strategy, the Agency normalized its software capability and migrated from twin Honeywell 6680s to an IBM 3081 environment. The characteristics of the normalization were as follows:

Project Initiation	-- April 1982
Project Completion	-- January 1983
System Implementation	-- 17 January 1983

Lines of Code	-- 200,000 COBOL
Terminals	-- 550
Transactions	-- 250,000 per day average

SOURCE COMPUTER	TARGET COMPUTER
H6680 (2)	IBM 3081
'68 COBOL	'74 COBOL
TDS	CICS
IDS	VSAM

The normalization process moved the Agency's capability from the source to the target environment, standardized the programs into ANS '74 COBOL, implemented information hiding of both the DBMS and TP functions, and documented the programs. Exhibit 13 shows the change brought about by normalization.

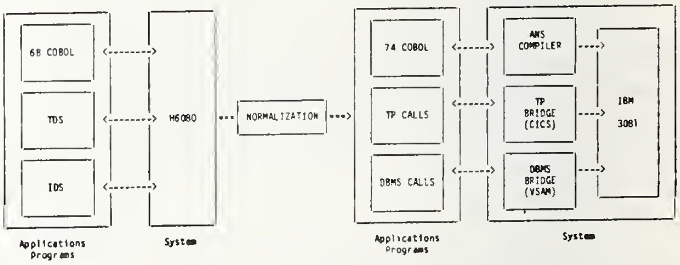


Exhibit 13. Example of Normalization

The Agency had three basic goals that had to be met for a successful conversion:

0 uninterrupted service during conversion;
0 reliable and maintainable information following conversion; and
0 conversion in a very tight time frame.

Automated normalization was selected as the migration strategy because it met these goals.

The Agency's Data Processing Center supports a nationally distributed network of terminals that is an integral part of the everyday operations of the total Agency and is critical to supporting the Agency's mission. It is essential that the computer provide uninterrupted and responsive service to the Agencies network of terminals. It was mandatory that this service be provided throughout the conversion process and that the uninterrupted responsiveness must not degrade during the transition period.

The Agency gathers, stores, tracks, and accounts for financial data representing billions of dollars per year and for personnel data covering hundreds-of-millions of people per year. This information must be accurate and auditable at all times. Therefore the software that gathers, processes and supports this information must be both reliable in its operations and maintainable as it is changed. It was mandatory that the conversion process must improve both the reliability and the maintainability of the software.

The Agency's schedule dictated that the conversion be completed in a nine month time frame, that parallel processing be limited to less than 30 days, and that the target environment be reliable and maintainable immediately after transition. It was mandatory that the conversion process be automated and meet this very tight schedule.

Exhibit 13 shows the before and after picture of the normalization. The system was normalized during the April to December period, was run in parallel for less than 30 days, and went live on January 17th 1983. The Agency met all three of its conversion goals. The Data Center provided uninterrupted service during transition, the software remained reliable and auditable, and the conversion was completed on time meeting the Agency's schedule.

5. Summary

Software improvement is a procedure that preserves an ADP organization's past investments and sunk costs in programs and information processing. Software improvement differs from software redesign or development because it minimizes the risks of reprogramming by modernizing in incremental, testable steps. Through software improvement, organizations can modernize their information processing in a controlled manner, thus minimizing risks.

Normalization is the initial step in software improvement; it is the standardization of existing software, making it more enhanceable, maintainable, and portable. Automated normalization is a cost-effective and resource-efficient technical fact, that has been successfully demonstrated in several instances and is available in today's marketplace. Automated normalization makes software improvement highly economical, feasible, and timely. In summary, it is a way to modernize existing systems while preserving past investments as much as possible, and putting the systems in a position to take advantage of new and emerging ADP technologies.

Algebraic Models for CPU Sizing

Robert A. Orchard

Department of Computer Science
College of Staten Island
City University of New York

ABSTRACT

This paper describes a CPU sizing methodology developed by the author for a corporate Performance and Configuration Group. The objective of this study was to effectively predict CPU utilization and total workload turnaround time for future batch workloads. This was accomplished through the implementation of certain algebraic models which successfully model the various components (i.e., CPU, I/O, etc.) of a computer system, capturing the dynamic interrelationship of hardware configuration, operating system logic and application workload. The result of this work is an algorithm which will accurately forecast average CPU utilization, volume independent CPU utilization, initiator turnaround and workload turnaround time for a given workload on various CPU models (3031, 168-3, 168-3 MP, 3033, 3033 MP). From a planning viewpoint, this information is extremely important in determining hardware needs as application workload characteristics change.

1. INTRODUCTION

This paper describes a CPU sizing methodology developed by the author for a corporate Performance and Configuration Group. The objective of this study was to effectively predict CPU utilization and total workload turnaround time for future batch workloads. This was accomplished through the implementation of certain algebraic models which successfully model the various components (i.e., CPU, I/O, etc.) of a computer system, capturing the dynamic interrelationship of hardware configuration, operating system logic and application workload. The result of this work is an algorithm which will accurately forecast average CPU utilization, volume independent CPU utilization, initiator turnaround and workload turnaround time for a given workload on various CPU models (3031, 168-3, 168-3 MP, 3033, 3033 MP).[1] From a planning viewpoint, this information is extremely important in determining hardware needs as application workload characteristics change. The model is formally developed in Appendix A and an overview and application of the model is presented in the following sections.

2. Model Overview

Input to the model consists of 'jobs' which are characterized by total problem program (pp) seconds consumed, I/O counts by device type (e.g., 3330, 3350, 3400, etc.,), total I/O count and initiator (priority group) assignment. From this input, the model predicts three system variables: CPU utilization, initiator turnaround and workload turnaround. Considering the complexity of a computer system which employs hardware and software interrupting, priority scheduling and system resource competition through the multi-programming of jobs this is a non-trivial task. The algebraic models implemented, however, are simple but effective in capturing the dynamics of such a computer system. For this model average I/O behavior is assumed (i.e., minimal I/O queueing).

[1] See Appendix B for the MIPS rates for the various CPU models.

The algebraic model approach views a workload as a queue of jobs competing for CPU time, where CPU time can be spent in problem program state (pp), supervisor state (ss) or wait state. The total number in queue is bounded by the total number of initiators active (multiprogramming level) and jobs are prioritized in order to keep the CPU as active as possible. The job initiated with the highest priority has the best chance of securing CPU time. CPU control is relinquished when I/O is requested by the job holding the CPU or if a higher priority job causes an interrupt. Once the I/O request is satisfied, the job again competes for CPU time. Viewed in this way, the amount of CPU time consumed for a job in a given period depends on the character of the competing jobs. Taking this one step further, it is possible to view this activity as a competition between initiator workloads rather than jobs. The character of an initiator is determined by the character of all jobs assigned to it. Each initiator is characterized by it's I/O rate[2] it's standalone CPU rate[2] and it's turnaround time[2]. The dynamic interrelationship of competing initiators is modeled through a set of simultaneous equations.

The number of simultaneous equations to model a given workload is the total number of initiators assigned to the workload. The solution of these equations determines which initiator drops out of the mix first since the "work" assigned to that initiator is complete. A new set of simultaneous equations for the remaining initiators and remaining "work" is then constructed. The solution to each set of simultaneous equations is the degraded CPU rates[2] for the active initiators in each time interval. Using these degraded CPU rates, the intervals for each initiator to complete their workloads are computed. The shortest interval identifies the initiator that drops out. The workloads for the other initiators are adjusted by the CPU time consumed by each initiator over the interval. The algorithm continues iteratively until all initiators complete their "work". Total workload turnaround then is the time it takes for the last initiator to complete its work measured from the start of the first initiator. CPU utilization for the machine under study is then calculated.

3. Data Preparation

An actual CPU sizing study was performed using workload information supplied by a large application on IBM mainframes. The workload characteristics (CPU problem program seconds and I/O (EXCP) counts by device type) were obtained using System Management Facility (SMF) log data collected on site. The objective of the study was to predict CPU utilization and workload turnaround for future projected workloads using the current batch workload as a baseline. Future workload characteristics were defined by the corporate Performance and Configuration Group.

An SMF Stripper program was used to extract the job characteristics used for input to the modeling program. The SMF stripper program stored the stripped data in a hierarchical database from which job and system summary reports were generated to help validate the models' predicted results.

4. Model Validation

For the purpose of validating the model, with respect to a baseline, one day was chosen from the monthly workload as input to the model program. The CPU utilization and workload turnaround for the test day was obtained via reports generated by the database system. Data for this day reflected heavy CPU and I/O activity. Selected jobs were extracted from the database, assigned to initiators and input to the model. The jobs were assigned to four initiators according to their I/O rate with highest priority given to I/O bound jobs. These assignments could have been made according to the actual initiators assigned to the baseline workload but this information was not available. However, the method used to assign jobs to various priority initiators by there IO boundedness (I/O rates) is in common use by computer system planners in charge of job scheduling. The idea, of course, being to overlap I/O and CPU utilization. The results, actual and predicted, are shown in Table 1.

Table 1

Validation Results

	Actual	*Predicted*	*% Error*
Problem program (pp) (minutes)	295.64	295.64	0.0
Supervisor state (ss) (minutes)	70.33	62.5	11.1
CPU utilization (percent)	55.4	51.1	7.3
Workload turnaround (hours)	11.0	11.68	5.8

[2] See Appendix A for definition.

It is important to note that the actual workload contained elements which we were not able to model and which were irrelevant to the batch workload modeled. These elements (e.g., IMS start-up and I/O from started tasks) were minimal but do inflate the observed error on supervisor state which we believe to be in the neighborhood of 3%-4% based on previous validation experiences with the model. Also, certain approximations to non critical operating system overhead in the model are "noisy".

5. Batch Workloads

Nine workloads were defined by the Performance and Configuration Group as input to the model. Table 2 briefly describes each of these workloads. Again, the objective of the study was to predict CPU utilization and turnaround time for each workload on various machines and to gain insight into the behavior of the workloads under machine upgrades.

Table 2

Batch Workloads

	Description
WORKLOAD A	anticipated workload for end of month processing
WORKLOAD B	anticipated workload for end of week processing
WORKLOAD C	anticipated workload for middle of week processing
WORKLOAD A1	subset of WORKLOAD A which could be split out and run on a separate processor (termed primary) in a dual computer environment
WORKLOAD A2	subset of WORKLOAD A (containing the remaining jobs of WORKLOAD A not included in WORKLOAD A1) which could be run on a second processor (termed support) in a dual computer environment
WORKLOAD B1	subset of WORKLOAD B for primary processor
WORKLOAD B2	subset of WORKLOAD B for support processor
WORKLOAD C1	subset of WORKLOAD C for primary processor
WORKLOAD C2	subset of WORKLOAD C for support processor

Each workload was run through the model for various machines (3031, 168-3, 168-3 MP, 3033, 3033 MP). Each workload run, utilized four initiators. For the purpose of comparison, the model results were divided into three groups as follows:

Group I - consisting of workloads A, B and C

Group II - consisting of workloads A1, B1 and C1

Group III - consisting of workloads A2, B2 and C2

The results from the various model runs were plotted by group and are shown in Figures 1-9. These graphs depict the workload turnaround times, the average CPU utilization and the volume independent CPU utilization forecast for the nine workloads modeled. Volume independent CPU utilization represents the CPU utilization when all initiators are competing for CPU time and indicates the CPU power required if the volume of work offered sustains all initiators active over an unspecified period of time. Individual initiator turnaround time were available, but were not plotted for this study.

6. Discussion

Perhaps the most interesting conclusion to be drawn from the model results, pertains to the character of the batch workload and its behavior under various CPU processing speeds. From the turnaround times forecast for the different workloads, the benefits realized by upgrading to a faster CPU (i.e., going from a 3031 to a 168-3, 3033, 3033 MP) became less and less. Looking at Figures 1, 4, 7 (Workload Turnaround) one can observe that the workload turnaround times predicted for a 3033 and 3033 MP are very close. The reason for this phenomenon is that the modeled workloads are becoming constrained by their level of I/O activity and the number of initiators assigned to the mix.

Another characteristic of this phenomenon is that in addition to the number of initiators assigned to the mix, the effectiveness of a particular assignment of jobs to initiator is *not* invariant under an increase in the speed of the CPU. For example, in comparing the workload turnaround time for workload A (Figure 1) with a subset of itself, workload A2 (Figure 7), the following facts are noted.

i. the assignment of jobs to initiators (the initiator structure) is different in Figure 1 than in Figure 7 but within each figure across machines the initiator structure is constant.

ii. on the 3031 the turnaround time for workload A is 48.5 hours and that for workload A_2 is 44.6 hours. On the 3033 MP the turnaround time for workload A is 13.8 hours while that of workload A_2 is 15.2 hours.

These facts suggest that as a workload becomes more I/O bound (i.e. faster CPU) the initiator structure may become a more predominant factor in turnaround time. The effect of initiator structure on turnaround time as a function of machine speed is being further investigated.

The above issues suggest that to insure batch workload turnaround improvement when upgrading to a faster CPU that the initiator structure should be studied. It may be necessary to change the number of initiators as well as the method of assignment of the jobs to the initiators. The optimum results will be a function dependency relationships between jobs in the workload. Further, as mentioned earlier, average I/O behavior was assumed throughout this study. Upgrading to a faster CPU will speed up the delivery rate of I/O requests to the I/O subsystems. This may cause I/O bottlenecks in the actual system. Such a situation may be relieved by a better distribution of data balancing across channels, controllers, and devices or by purchase of additional peripheral hardware.

7. Summary

An algebraic model for a multiprogrammed computer system has been developed (Appendix A) and an application of the model to a problem in CPU sizing described. A more extensive algebraic model incorporating other sources of contention (I/O subsystem, memory, etc.) has been developed and will appear in a forthcoming paper. Algebraic models other than simultaneous linear equations are available. I would like to take this opportunity to acknowledge many early stimulating discussions with H. Pat Artis on dynamic job scheduling, which motivated this research on algebraic models of computer systems.

References

[1] H. P. Artis, "Capacity Planning for MVS Computer Systems", in *Performance of Computer Installations*, D. Ferrari, Ed., North Holland Publishing, 1978.

[2] P. J. Denning, J. P. Buzen, "Operational Analysis of Queueing Network Models", *Computing Surveys*, Vol. 10, No. 3, Sept. 1978.

APPENDIX A

An Algebraic Model for CPU Sizing

A.1 Introduction

A new methodology for modeling computer systems has been developed using sequences of algebraic models. The approach can be used in conjunction with dynamic mix analysis techniques on workload clusters[1] and is similar in flavor to operational models of computer systems[2]. The *particular* algebraic model used in this paper for CPU sizing will be introduced as a set of simultaneous equations. The modeling approach taken here consists of the following mainpoints:

a. The conversion of application workload into workload on the various components of the system.

b. The rates at which this work is carried out can be represented by the utilizations of the components involved (CPU, channels, devices, etc.).

c. By studying the logic of the operating system, *critical sequences* of highly repetitive operating system module activity can be isolated. These are primarily sequences necessary to support the read and write activity of an application program and include as elements such modules as the I/O supervisor, interrupt handler, and dispatcher.

This information, together with certain application program parameters, is sufficient to determine the induced work and work rates (utilizations) on all processes classically of interest in the system. For purpose of *this paper* attention is restricted to CPU utilization only.

In this model it is assumed that no I/O contention (queueing) is occurring and that all I/O operations take the "average" time to complete. A parameter to account for deviations from this assumption is built into the model for potential use. It is also assumed that paging (or swapping) is not a significant problem and in a similar manner as I/O degradation, a parameter to account for deviations from this assumption can be built into the model. Submodels can then be used to estimate these parameters. This expanded approach has been used and benchmark validated in a more complete algebraic model of a computer system than outlined in this paper.

We also assume there are N jobs competing for the central processing unit (CPU) under a preemptive priority dispatching discipline for service. An application program in control of the CPU can be interrupted and lose control of the CPU if either a program of higher priority preempts or the program itself issues a request for an I/O operation.

We are interested in isolating certain key parameters of application programs which characterize the logical interaction between program and operating system modules. These parameters, ideally, will be invariant under multiprogramming since they will represent interaction with the operating system which must be carried out regardless of the collection of programs with which a given program is executed. Total number of application program CPU seconds consumed and the number of I/O's issued per application program CPU second are examples of such invariant parameters. As we will see, the latter will allow us to calculate the CPU time expended by the operating system to service the application program requests for I/O operations as a function of the multiprogrammed mix.

The following is a list of operational definitions necessary to describe the quantities used in the model. For a given application program, P_i, the following standalone attributes are defined.

C_i = total application problem program CPU time for program i

T_i = total single thread elapsed time of program i

E_i^t = total number of tape I/O's issued by program i

E_i^d = total number of disk I/O's issued by program i

e_i = $(E_i^t + E_i^d)/C_i$ is called the I/O rate of program i

r_i = C_i/T_i is called the CPU rate of program i

The following multiprogrammed attributes are also defined.

$\hat{T}_i$ = total elapsed time of program i when multiprogrammed

$\hat{r}_i$ = C_i/T_i is called the degraded CPU rate for program i

Δt_i^t = average time to do a tape I/O

Δt_i^d = average time to do a disk I/O

Certain quantities related to the operating system will also be referenced as follows.

θ' = sum of CPU timings through all operating system modules invoked to support *one* I/O. The sequence of operating system modules invoked is called the I/O *critical path.*

N = number of programs running in the multiprogrammed job mix (i.e., number of initiators active).

$\theta(N)$ = CPU rate of all operating system module activity not on the I/O critical path. This is a function of N and the multiprogrammed mix characteristics.

A.2 Model Formulation

A.2.1 Operating System If we assume the operating system modules run in supervisor state disabled for interrupts, then the operating system modules once they gain control of the CPU cannot be preempted by an application program. Therefore, operating system modules on the I/O critical path invoked by the process of issuing an I/O request, take on the priority of the invoking programs in order to gain control of the CPU, but once in control, they behave as the highest priority jobs in the multiprogrammed mix.

Since an operating system module is itself a program, the CPU rate of the module is proportional to the CPU rates and I/O rates of the programs invoking it. Once activated, the module will degrade all application program CPU rates with the exception of the invoking program. All operating system module activity not included in the I/O critical path is lumped into one pseudo-operating system module which is considered to run at highest priority with CPU rate $\theta(N)$.

The CPU rate of that portion of the operating system representing modules on the I/O critical path is given by

$$I = \sum_{i=1}^{N} \hat{r}_i \, e_i \, \theta'$$

where

$\hat{r}_i$ = CPU seconds/elapsed second

e_i = I/O's/CPU second

θ' = CPU seconds/ I/O

Note that $\hat{r}_i$ is the actual degraded CPU rate of program i which we will ultimately solve for.

If S is the total operating system CPU rate then

$$S = \theta(N) + I$$

In order to fully characterize the total operating system CPU rate, it is necessary to determine $\theta(N)$. Total CPU rate of all I/O critical operating system modules is obtainable as well as total operating system CPU rate. Hence, we may assume both S and I are known quantitatively. Unpublished experimental results with operating systems have indicated that $\theta(N)$ varies linearly with increasing multiprogramming level. We therefore assume

$$\theta(N) = k(N)S$$

where k(N) is a linear function depending on the number of programs in the mix. Since S is $\theta(N) + I$,

$$\theta(N) = k(N)(\theta(N) + I)$$

or

$$\theta(N) = I(k(N)/(1-k(N))$$

$\theta(N)$ is now seen to be a function of multiprogram depth and though the expression I also a function of the I/O characteristics of the programs being multiprogrammed. The expression k(N)/(1-k(N)) will be referred to as simply the term K, the dependence on N being assumed. k(N) can be empirically determined given θ'), total number of I/O's in a mix, multiprogramming level, S and the discussion in this section.

A.2.2 Application Programs Next we turn our attention to determining the degraded CPU rates $\hat{r}_i$ of each program in the mix in order to determine total CPU utilization required and the turnaround time for the mix.

Essential to the modeling technique developed in this paper is a three level hierarchical view of time. Given any process, one considers first the passage of elapsed (clock on the wall) time. Within an interval of such time, certain subintervals may be made available within which a process can become active. That is, the process may become active for none, some, or all of the subinterval. Hence, one can distinguish three *types of time*: elapsed time, available time, and process active time. This situation is indicated in Figure A-1.

THREE LEVEL VIEW OF TIME

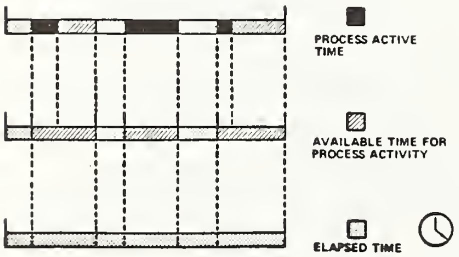


Figure A-1

Note that if there are several processes competing for available time within an interval of elapsed time, then what is nonavailable time for one process may be available time for competing processes. This is true for example when programs compete for the elapsed time of a CPU. On the top line of Figure A-1, the dotted regions represent time spent on competing processes and the hashed region time that was available to a process but not used, i.e., the process was not active. The darkened regions indicate process active time. There are therefore, two possible ways to represent a process utilization; with respect to elapsed time and with respect to available time. This distinction is of fundamental importance in the solution of many computer system problems since any attempt to increase a process utilization beyond the maximum value indicated when utilization is computed with respect to available time must be achieved at the expense of competing process utilization.

If program execution in a multiprogrammed computer system is viewed as a process competing for available CPU time, then the degraded CPU rate of a program, as the model will calculate it, is equivalent to process utilization with respect to elapsed time and not available time. Its single thread CPU rate is equivalent to process utilization with respect to available time. For the single thread (standalone) case, available time is precisely elapsed time. The degraded CPU rate for program i, $\hat{r}_1$, is unknown since $\hat{T}_i$, the degraded elapsed time, is not known. However, $\hat{r}_1$ may be represented as an analytic expression derived as follows.

Let us assume that program i has higher priority than program i + 1 and that the operating system modules, once they have gained control of the CPU, cannot be preempted. Further, suppose a unit of available CPU time is made accessible to programs in a multiprogrammed mix. We would like to characterize the "average" competition for the unit of available CPU time by the programs and system modules constituting the mix. In other words, given a unit of CPU time (e.g., a CPU second), it is of interest to determine how the time is expended on the various processes competing for the CPU resource.

Consider an available CPU second. Since the operating system has top priority, $\theta(N) + I$ is the average fraction of the second taken by the operating system. The CPU time remaining for problem program state activity is then $1 - (\theta(N) + I)$ seconds. For a particular program, the expression representing operating system overhead time unavailable for problem program activity must be modified slightly. A program will never be degraded by its own I/O operating system overhead since it is assumed that it is no longer competing for the CPU resource until the I/O request is satisfied. Hence, define

$$I_i = I - \hat{r}_i e_i \theta'$$

so that I_i, represents all critical operating system module CPU rate except that induced by program i.

Assuming that no I/O contention is present, the top priority application program will take, on the average, a fraction r_1 of the available CPU time or

$$\hat{r}_1 = r_1(1 - (\theta(N) + I_1))$$

of the initial available CPU second. The next highest priority program will take on the average a fraction r_2 of the remaining CPU time or

$$\hat{r}_2 = r_2(1 - (\theta(N) + I_2 + \hat{r}_1))$$

of the initial available CPU second.

In general, the ith program experiencing no I/O contention will take on the average a fraction r_i, of the remaining available CPU time or

$$\hat{r}_i = r_i(1 - (\theta(N) + I_i + \sum_{j<i} \hat{r}_j))$$

of the initial available CPU second. Figure A-2 indicates the initial available CPU second and the corresponding time lines for the equations given above.

If it is assumed that I/O contention is present, then the above equations can be modified so that,

$$\hat{r}_i = d_i r_i(1 - \theta(N) + I_i + \sum_{j<i} \hat{r}_j)) \qquad \text{(A-1)}$$

$$i = 1,2,...N$$

where d_i is a parameter reflecting degradation due to I/O contention and $0 \leq d_i \leq 1$. d_i may itself be represented as an algebraic model or as the output of a simulation or queueing model.

r_i may be approximated as follows,

$$r_i = C_i/(C_i + (E_i^t \Delta t_i^t + E_i^d \Delta t_i^d) \qquad \text{(A-2)}$$

$$+ \theta'(E_i^t + E_i^d))$$

or obtained by actually running the job on the computer system.

Equation (A-1) represents a set of N simultaneous equations in the N unknowns, $\hat{r}_i$. Solving this set of equations for the $\hat{r}_i$, with $d_i = 1$ for all i, allows us to them compute the following functions of the degraded rates.

Turnaround (Service, Response) Time for Program i

$$\hat{T}_i = \frac{C_i}{\hat{r}_i}$$

Supervisor State (SS) CPU Utilization

$$S = \theta(N) + I$$

$$= \sum_{i=1}^{N} (K + 1)e_i\theta'\hat{r}_i$$

Problem Program (PP) CPU Utilization

$$P = \sum_{i=1}^{N} \hat{r}_i$$

Total CPU Utilization

$$U = S + P$$

$$= \sum_{i=1}^{N} (1 + (K + 1)e_i\theta')\hat{r}_i$$

Contribution of each Program to Total CPU Utilization

$$\underset{PP}{\hat{r}_i} \quad + \quad \underset{SS}{(K + 1)e_i\theta'\hat{r}_i}$$

APPENDIX B

MIPS RATES

3031	1.2
168-3	2.7
168-3MP/AP	4.5
3033	4.6
3033 MP/AP	7.6

INITIAL AVAILABLE CPU SECOND

OPERATING SYSTEM	AVAILABLE CPU TIME FOR PROGRAM 1
$\theta(N)+I_1$	$1-(\theta(N)+I_1)$

OPERATING SYSTEM	PROGRAM 1	AVAILABLE CPU TIME FOR PROGRAM 2
$\theta(N)+I_2$	$r_1(1-(\theta(N)+I_1))$	$1-(\theta(N)+I_2+\hat{r}_1)$
	$\hat{r}_1$	

OPERATING SYSTEM	PROGRAM 1	. . .	PROGRAM i	AVAILABLE CPU TIME FOR PROGRAM i+1
$\theta(N)+I_{i+1}$	$r_1(1-(\theta(N)+I_i))$	. . .	$r_i(1-(\theta(N)+I_i+\sum_{j<i}\hat{r}_j))$	$1-(\theta(N)+I_{i+1}+\sum_{j<i+1}\hat{r}_j)$
	$\hat{r}_1$		$\hat{r}_i$	

Figure (A-2)

Workload Turnaround (hrs) - Group I

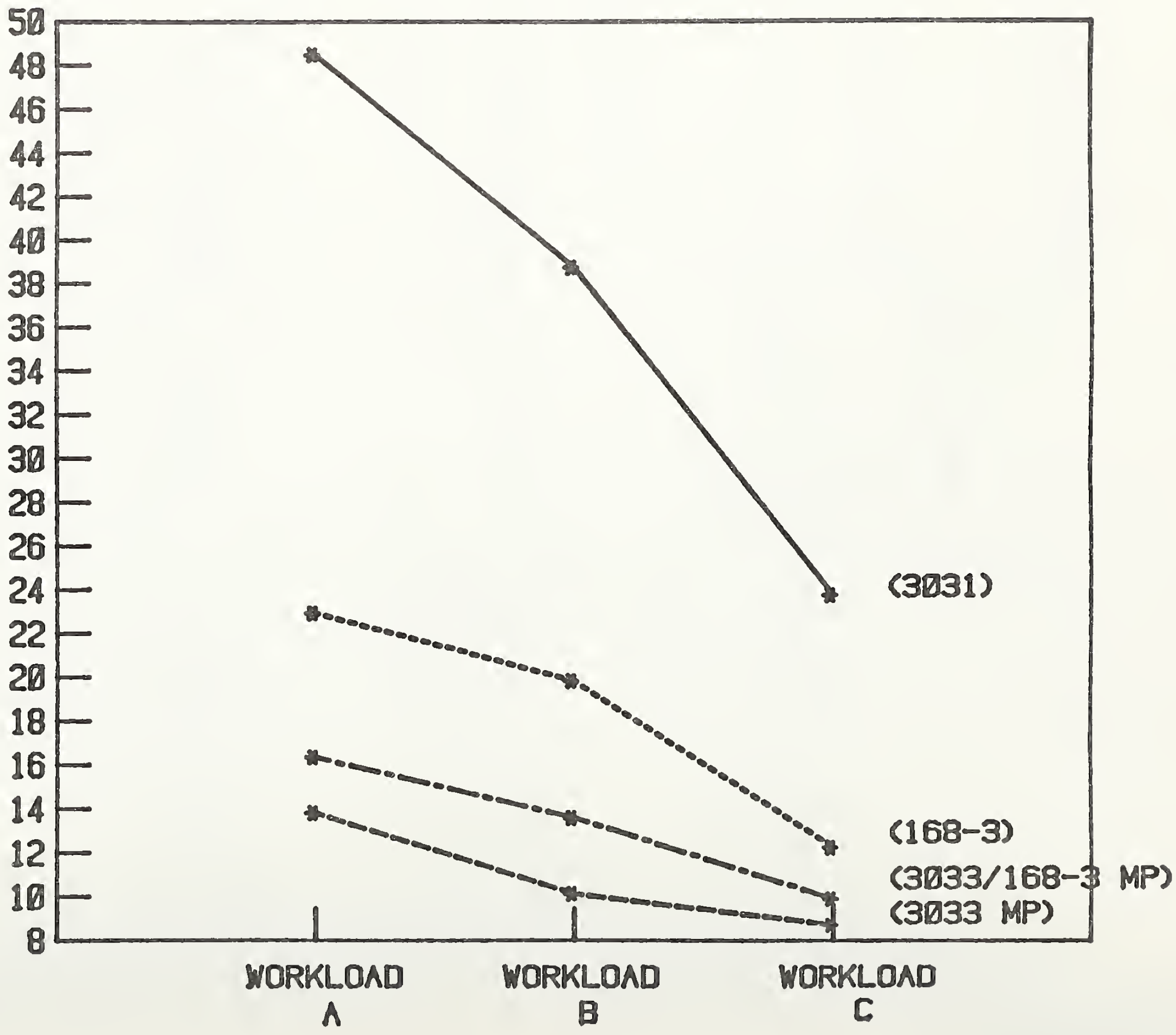
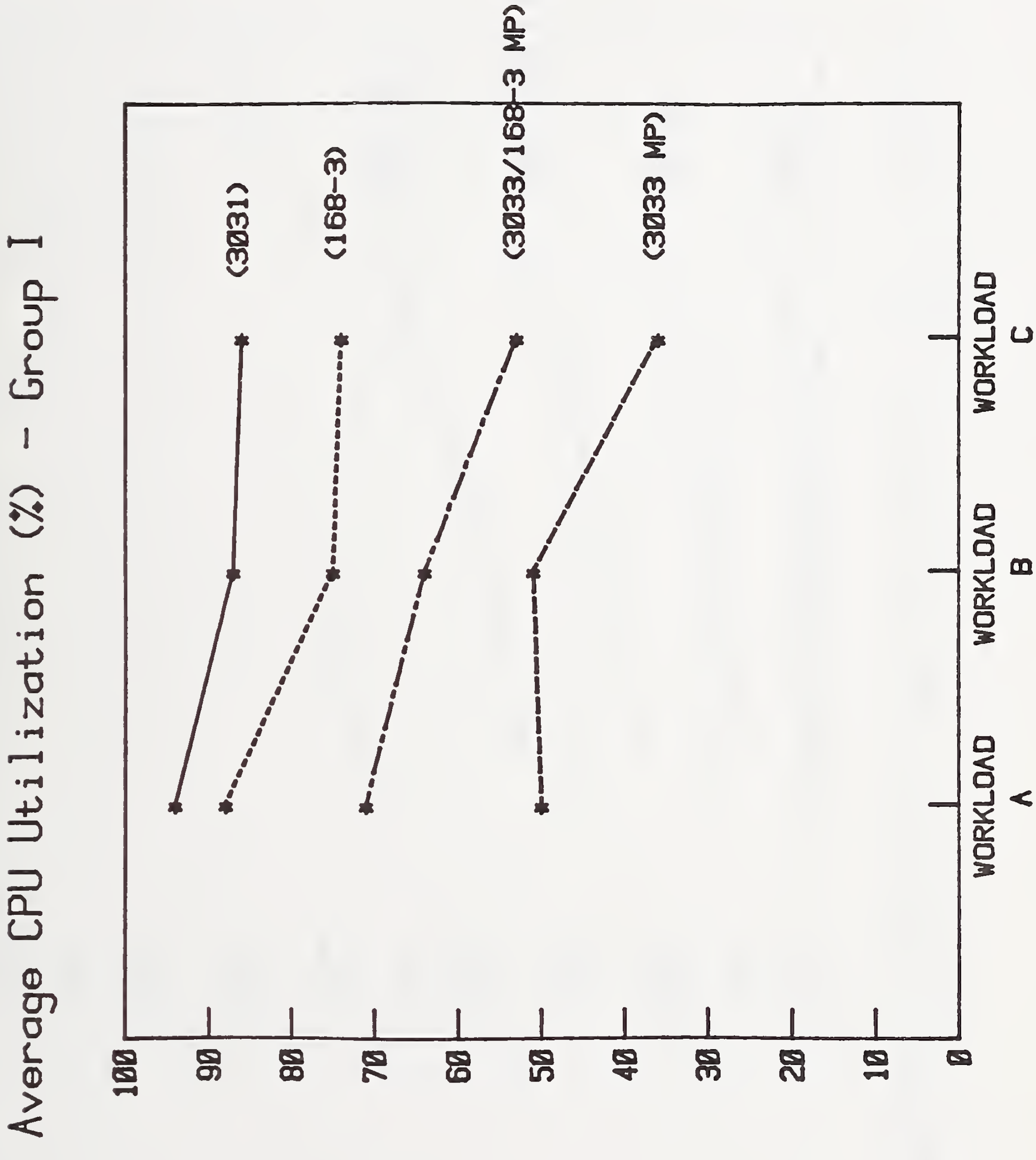


Figure 1


Average CPU Utilization (%) - Group I
100
90
80
70
60
50
40
30
20
10
0
WORKLOAD A
WORKLOAD B
WORKLOAD C
(3031)
(168-3)
(3033/168-3 MP)
(3033 MP)

Fiqure 2

Volume Independent CPU Utilization (%) - Group I

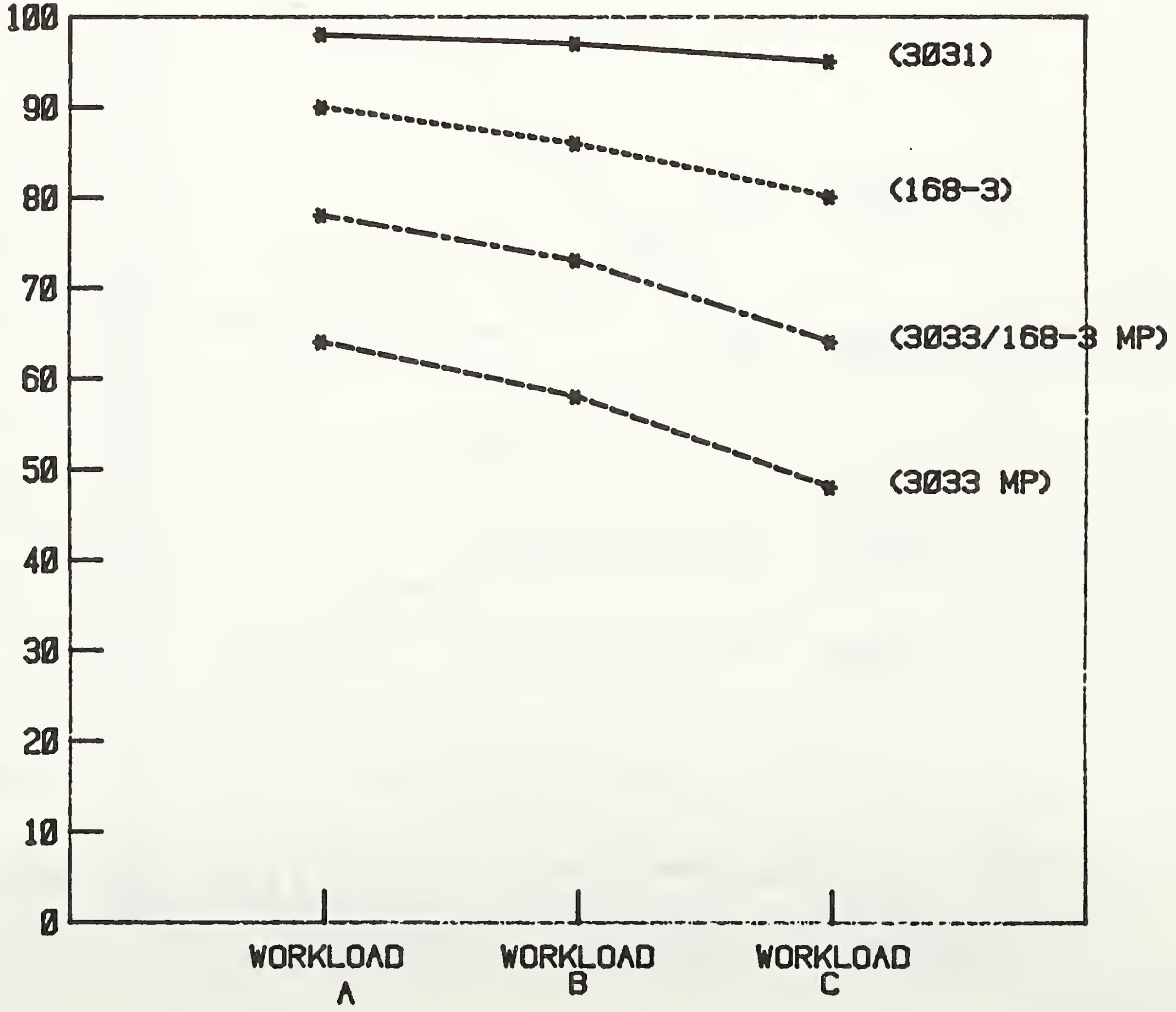
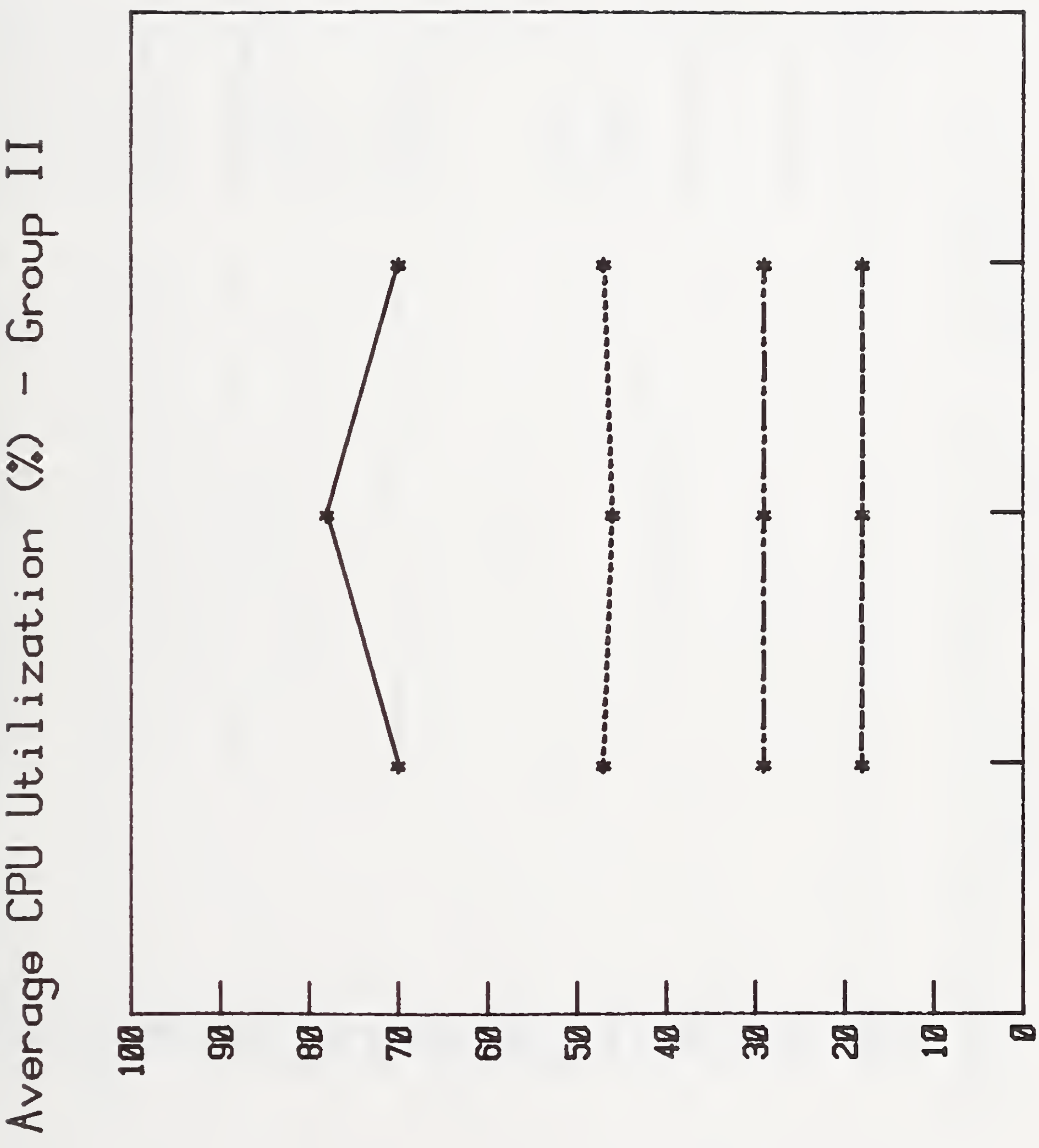
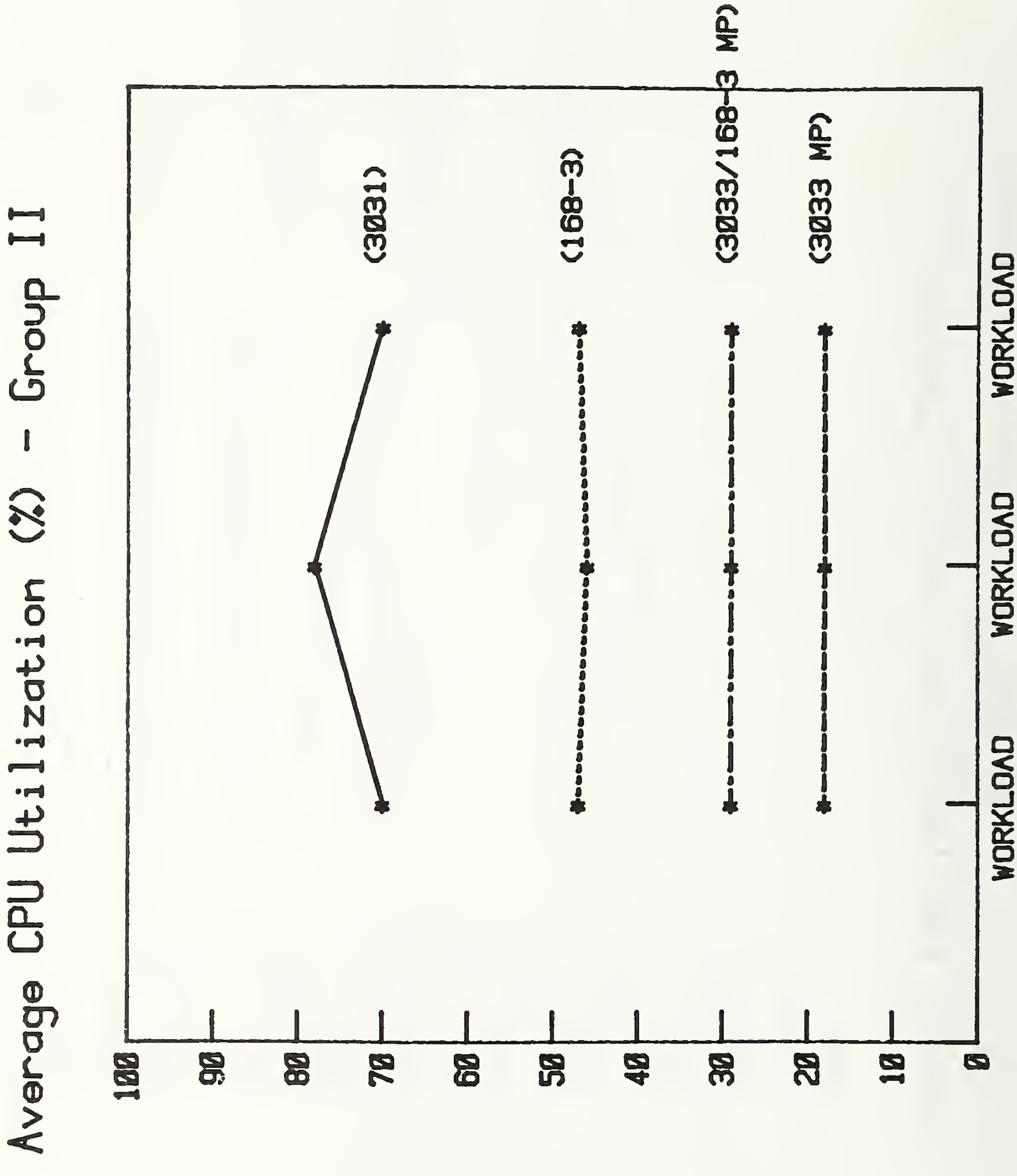


Figure 3


Average CPU Utilization (%) - Group II
100
90
80
70
60
50
40
30
20
10
0

Fiaure 4


Average CPU Utilization (%) - Group II
100
90
80
70
60
50
40
30
20
10
0
WORKLOAD A1
WORKLOAD B1
WORKLOAD C1
(3031)
(168-3)
(3033/168-3 MP)
(3033 MP)

Fiqure 5

Volume Independent CPU Utilization (%) - Group II

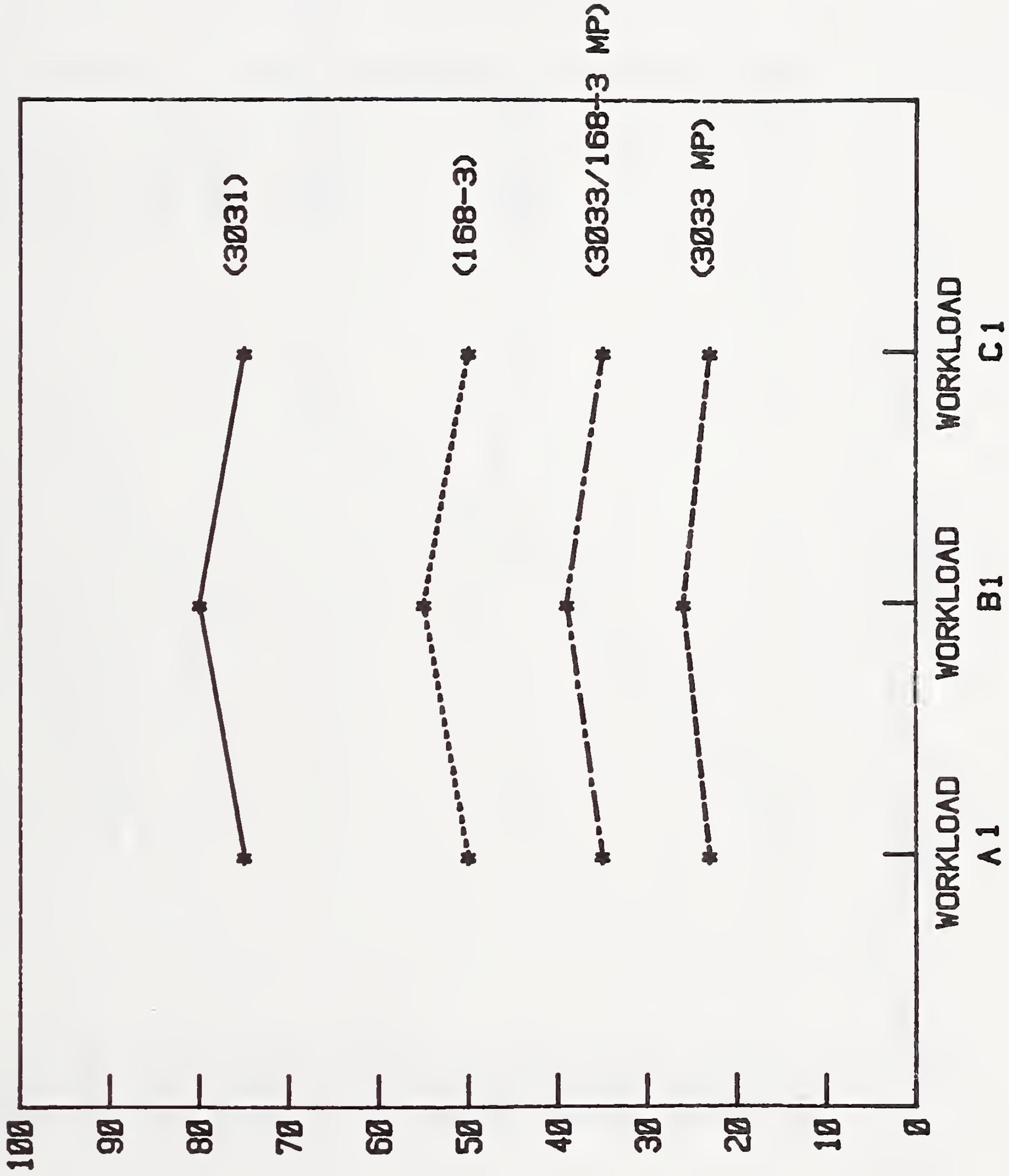
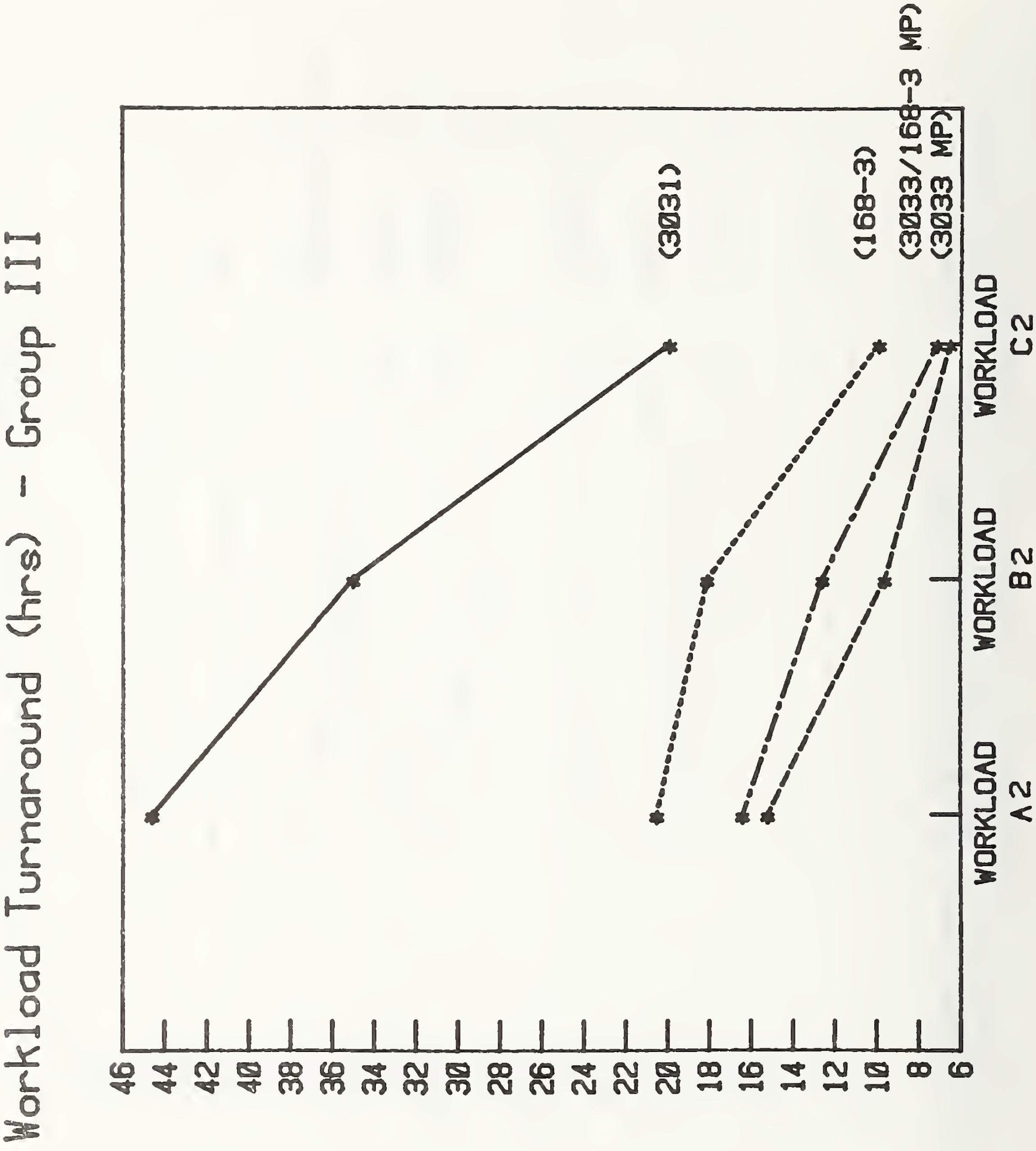
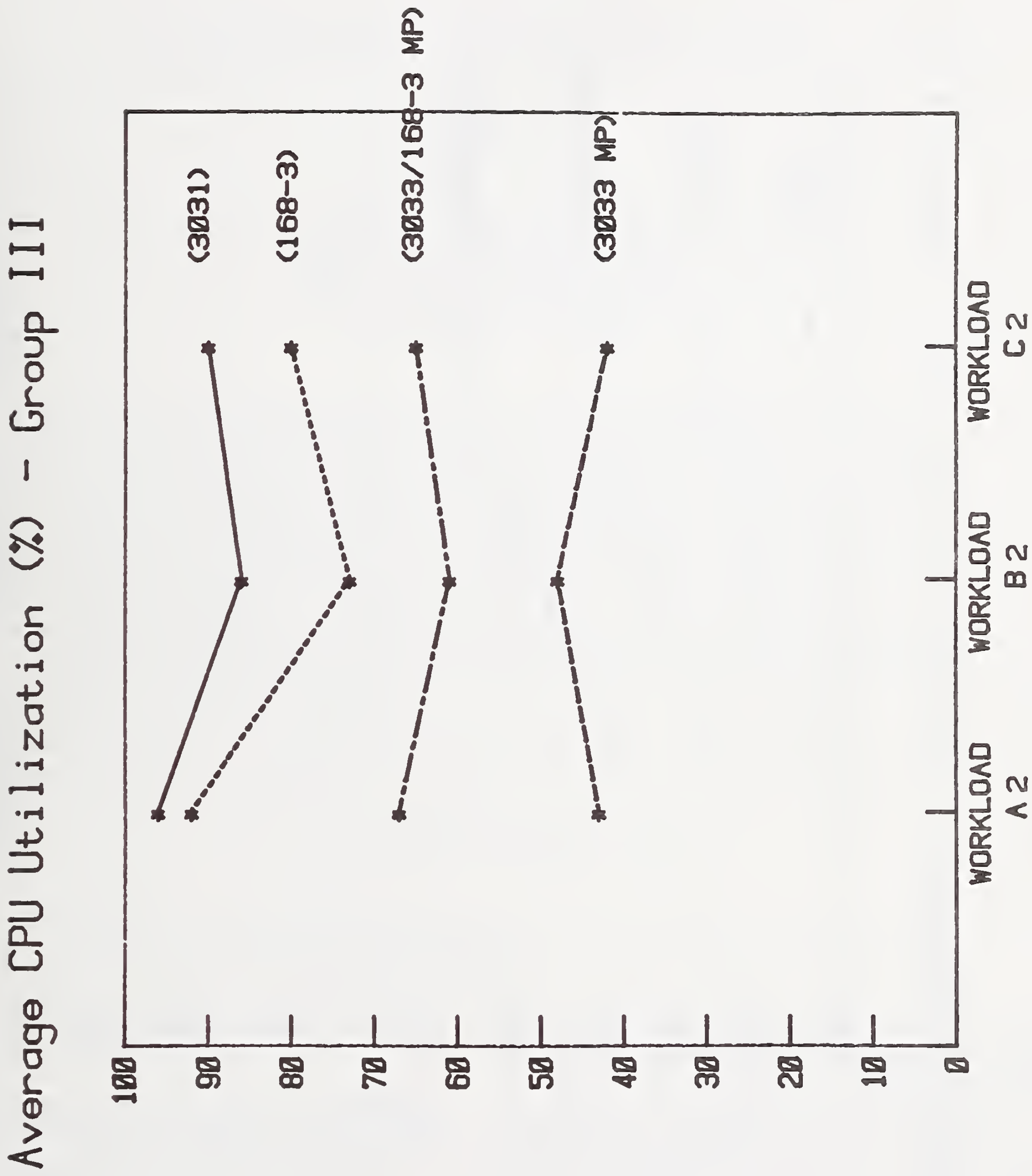


Figure 6


Workload Turnaround (hrs) - Group III
46
44
42
40
38
36
34
32
30
28
26
24
22
20
18
16
14
12
10
8
6
WORKLOAD A2
WORKLOAD B2
WORKLOAD C2
(3031)
(168-3)
(3033/168-3 MP)
(3033 MP)

Fiqure 7


Average CPU Utilization (%) - Group III
100
90
80
70
60
50
40
30
20
10
0
WORKLOAD A2
WORKLOAD B2
WORKLOAD C2
(3031)
(168-3)
(3033/168-3 MP)
(3033 MP)

Fiqure 8

Volume Independent CPU Utilization (%) - Group III

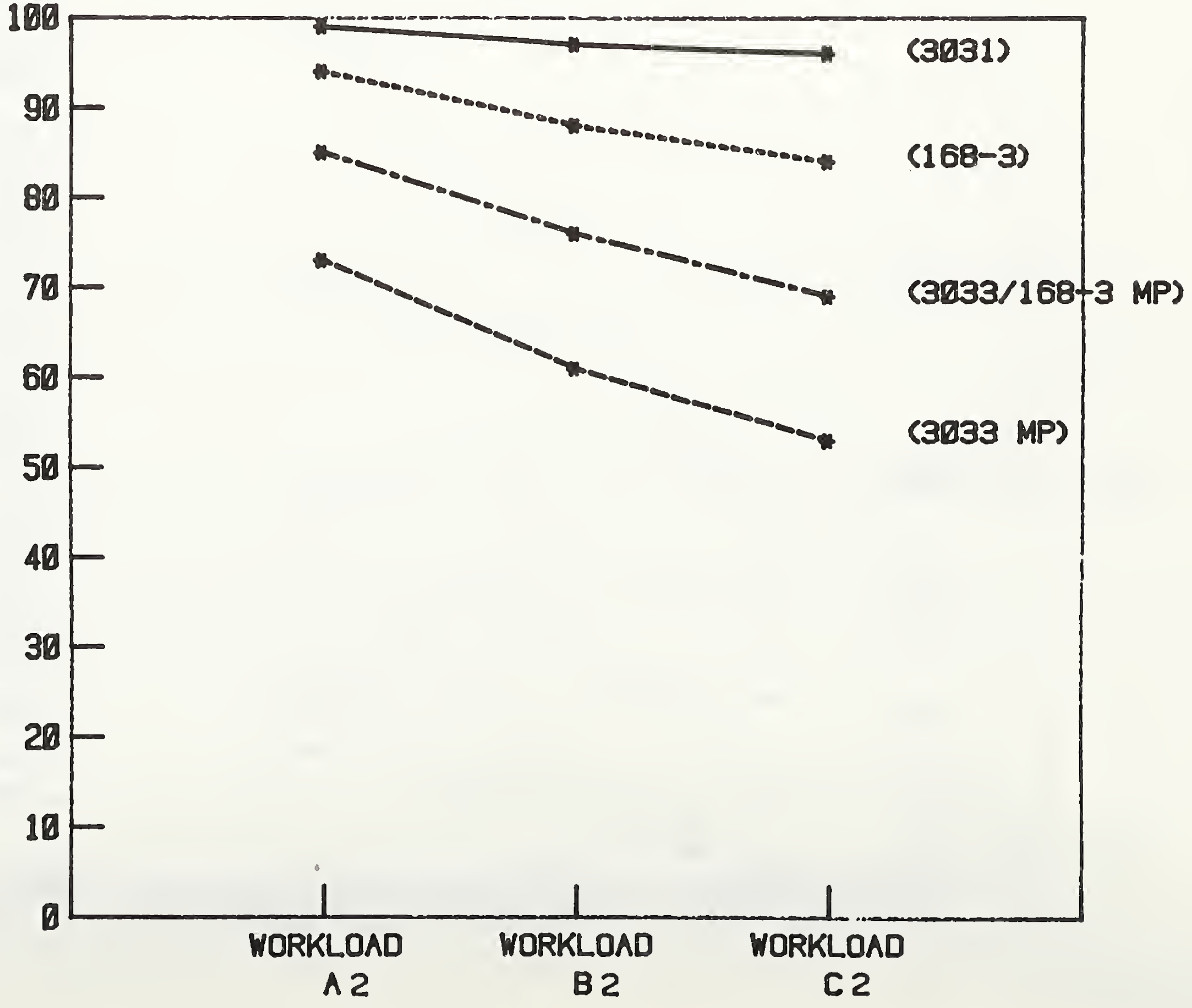


Figure 9

ESTABLISHING A SOFTWARE ENGINEERING TECHNOLOGY (SET)

L. Arnold Johnson
and
William R. Milligan

Federal Software Testing Center
Office of Software Development and Information Technology
General Services Administration
5203 Leesburg Pike, Suite 1100
Falls Church, VA 22041

This paper will discuss the concept of software engineering as provided by Dr. Barry Boehm as "the application of science and mathematics by which the capabilities of computer equipment are made useful to man via computer programs, procedures, and associated documentation." It can serve as a starting point for developing and institutionalizing a modern Software Engineering Technology (SET). The document defines key elements which comprise a SET and suggests a method for formulation of these elements into a technology that encompasses all of the primary stages of the software life cycle. It presents a discussion of the types modern day software engineering practices. The approach emphasizes the incremental integration of software tools into the technology as a means of increasing productivity, establishing regularity and uniformity, and improving control over software systems.

"The past is but the beginning of a beginning, and all that is and has been is but the twilight of the dawn."

H.G. Wells
The Discovery of the Future (1901)

Key Words: Software Engineering Technology (SET); Software Engineering; software tools; software management.

1.1 Software Engineering

Software engineering encompasses a wide range of techniques and methods for managing, developing, and maintaining computer software. It is sometimes thought of in the more restrictive sense to cover only programming methodology. Software engineering actually covers a much broader scope and includes all of the disciplines which are used in dealing with software throughout its life cycle.

A more formal definition of software engineering is provided by Dr. Barry Boehm as "the application of science and mathematics by which the capabilities of computer equipment are made useful to man via computer programs, procedures, and associated documentation" [1]. Software engineering is sometimes referred to as the discipline which brings order to the software life cycle development process. There are many techniques such as top down design, structured programming, thread testing, HIPO charts, etc., all of which bring a form of discipline to the development and management of software. Defining the techniques employed by an organization through the use of standards and procedures is a primary means of establishing order in software development and management.

Along with a discipline come the measurements and controls which permit software to be evaluated and subsequently managed. Activities such as critical design reviews, program design reviews, and software change controls usually accompany software engineering methodologies. These functions are necessary in order to determine if the software meets the needs of the end user and is easy to maintain. A Software Engineering Technology (SET) includes the development of software standards, development of computer and manual procedures, acquisition of the necessary software tools, and integration of these elements into a suitable machine environment. The aim of the SET is that it be a practitioner's approach to software engineering [2]. In doing so the technology must be flexible, yet should not sacrifice principles. A comprehensive SET for software development must consider all aspects of the software life cycle if desired results are to be obtained. For this to be possible, the factors which affect the software life cycle phases must be identified and a formal technology established.

1.2 Components of a SET

A SET consists of two major components - stages and elements. Stages are discrete phases of the software life cycle identified by the type of activities associated within the stage. The six stages of the software life cycle for a SET are-

- requirements definition and analysis;
- design;
- programming;
- validation;
- operation; and
- review.

Elements of a SET are the principle factors which direct and control the software activities within each software stage. The five elements for a SET are-

- standards;
- procedures;
- tools;
- quality assurance; and
- training.

The combination of these five elements within a particular environment over the six stages of the software life cycle constitute a SET. This SET can be built in such a way that resources of varied quality can be used to accomplish similar engineering tasks. Without any of the five elements, an engineering project will not produce a technology, but rather an unused set of tools, standards, and procedures [3].

The association of the five different elements with the six stages is viewed as a two dimensional matrix as illustrated in Figure 1. The blocks of the matrix, which are the intersections of elements within a stage, represent the theoretical details of the technology. After the SET is developed, these blocks should contain the detailed documentation for the standards, procedures, tools, quality assurance, and training for each stage. The lines dissecting the stages and elements provide a natural boundry for the SET components. These divisional boundaries are logical control points in the technology from which reviews can be performed, quality ascertained, errors identified and corrected, and management decisions made. Also, the vertical lines dividing the stages are logical points for products, deliverables or milestones to occur. In addition, the system environment (hardware, methods of operation, etc.) provide a third unillustrated, dimension to the matrix. As an organization manages or changes its data processing environment, the matrix may have to be modified accordingly as additional tools, procedures, and standards enter the environment.

1.3 SET Iterative Review Process

Because of the numerous directions software can take, software control becomes a process of iterative reviews. All components within the software life cycle need to be checked and rechecked to ensure that none of the meaning or usefulness desired from the software is lost. One method, as reflected in this SET, is to continually test the quality of products from each stage in the software life cycle.

1.4 Key Concepts of a SET

The foundation for a SET consists of the objectives to be achieved from a technology. The building blocks for meeting these objectives are the standards, procedures, tools, quality assurance, and training elements available in an appropriate environment. For the technology to be effective, these elements must be interrelated so that each element compliments the other.

Two of the most important concepts for making effective use of any SET are the controls administered and management support for the SET program. Management must be willing to allocate a sufficient, but reasonable, amount of money, personnel, time, and other resources to the SET program; not only to develop it, but also to maintain and control it.

1.5 SET Objectives

No organization or technology would have justification for existing unless identified benefits were realized. This is particularly true for a software engineering technology.

	STAGES OF SET					
ELEMENTS OF SET	REQUIREMENTS DEFINITION & ANALYSIS	DESIGN	PROGRAMMING	VALIDATION	OPERATIONS	REVIEW
STANDARDS AND GUIDELINES						
PROCEDURES						
TOOLS						
QUALITY ASSURANCE						
TRAINING						

Figure 1: Component Matrix of the SET

Thus, it is paramount that defined objectives be in place so there is a common goal upon which a SET can be built. Primary objectives of the SET are to-

- reduce overall cost of software by reducing the effort spent on maintenance through better planning, design, testing, and control of software resources;
- improve software reliability by increasing the degree of software correctness, minimizing uncertainty, maximizing data integrity, and improving system security;
- improve maintainability of software by making software structure consistent, isolating functions, having up-to-date documentation, facilitating program understanding, standardizing interfaces, managing mechanisms to control and apply changes or enhancements quickly, etc.;
- improve portability by isolating architecture dependencies, reducing or eliminating operator intervention, eliminating nonstandard source code, using high-level computer languages, and reusing source code (i.e. reducing development redundancy);
- ensure software is useable by making sure it meets requirements and maximizes error detection in the early stages of software development;
- obtain regularity and uniformity in such areas as program format, naming conventions, programming standards, system design methodology, data isolation, maintenance change methodology, function interchangeability, single-source maintenance, and test and acceptance criteria;
- establish controls and measurement of software for project monitoring, evaluation of software, management decisions, tracking, auditing, etc.;
- improve organization productivity through increased programmer productivity, increased hardware performance, reduced learning curves, increased programmer effectiveness, and broader skills at both technical and management levels; and
- improve product quality and responsiveness to user requirements.

1.6 Environment's Effect on the SET

There are certain factors which have an effect or influence upon the development and implementation of a SET. This common influence on the SET is referred to as the environment. The environment is the aggregate of organizational, technical, and managerial conditions of the ADP activity, which make the definition of each SET unique. Thus, the effect the environment has upon the SET is different for each organization. Add to this the fact that an organization's environment is constantly changing with time. Therefore, the SET must be flexible and conducive to changes in the environment.

2. Overview of Software Life Cycle Stages

The software life cycle includes the six stages of-

- requirements definition and analysis;
- design;
- programming;
- validation;
- operation; and
- review.

2.1 Requirements Definition and Analysis Stage

The Requirements Definition and Analysis Stage begins after a feasibility study has recommended that a new or modified system be developed for computer processing, and that recommendation has been accepted by management. The purpose of this stage is to define specific functional capabilities required for the system, define performance requirements that must be met, and identify all information that the system will use or produce. The results of this stage become input to the next stage, which is the design of the new or modified system. To avoid limiting the system designer's ability to consider a variety of design options, requirements defined by this stage must concentrate on specifying what the system is to do -- not how to do it.

Analysis includes the study of a business area or application, leading to the specification of a new or modified system. It consists of interviewing the user about what the current system does or should do, what extra features are desired in the new system, and what constraints should be placed on the new system. There are repeated interactions with the user in order to reach a clear understanding of new system requirements. The most important product of system analysis is the functional specification. The user must be given adequate time to review the functional specification before it is passed to the system designer, as this can alleviate problems later. If care is taken to ensure that all of the data and processing requirements have been identified, then required user changes can be kept to a minium. However, even though care is taken to review and finalize requirements with the user, the definition of requirements is an interactive process throughout the system life cycle, with specification changes identified during later life cycle stages.

2.2 Design Stage

Design includes the tasks of detailed specification of software, data, hardware and processing requirements, and the segmentation of processes into programs. The design process involves moving through successive levels until the system has been defined sufficiently for software development to begin. This decomposition facilitates human comprehension by breaking down a large, complex problem into smaller, more manageable pieces.

In addition to moving through successive levels of conceptualization in decomposition of the problem, there is also some iteration of the steps in the design process. Most often the initial solution to the problem is not the best; and as the designer moves from one level of conceptualization to the next, more insight is gained into the ramifications of the problem and design refinements may then be made.

A system can be viewed as a group of data processing entities. The designer describes these entities in terms of their input/output and processing performed to accomplish the transformation from input to output. Interfaces and control sequences of the data processing entities are described along with descriptions of the data that flows between them. This forms the "logical" structure of the system.

2.3 Programming Stage

Programming includes detailed design of the processing logic, preparation of a plan for testing the program, development of test data, code production, unit testing the code, and documentation. This stage begins with the receipt of a specification for the development of a new program or the modification of an existing program. The programmer's first act is to obtain a good definition of the problem to be solved. He analyzes requirements, asks questions if clarification is needed, and recommends modifications to the program specification if it contains errors, omissions, or obvious inconsistencies.

Once the problem is thoroughly defined, detailed program design begins. The program design is complete when the program specification, describing program construction, is finished. To predict whether the designed program will function as intended, designs should be verified by someone other than the designer as the designer cannot always objectively evaluate his own work.

After program design, the programmer codes the program. In some instances design may not be completely finished before coding begins. Modern programming techniques such as modularization and top down development, permit flexibility in scheduling program development activities by separating the program into logically distinct portions which can be developed and tested independently. As a result, there is a great deal of overlap of design, coding, testing, and documentation activity throughout the Programming stage.

Unit testing follows coding. This purpose of unit testing is to ensure that the unit, or program, released for system testing is free from internal logic or format errors and conforms to its specifications.

2.4 Validation Stage

Validation encompasses system testing followed by acceptance testing. The purpose of system testing (sometimes referred to as integration testing) is not to retest all of the detailed functions within each program since that would unnecessarily duplicate unit testing. Instead, its purpose is to connect the program units to determine whether they function together in tandem. Thus, the main testing emphasis is on the interaction between and interoperability of software components and their interfaces.

Planning for system testing begins in the earlier stages of the software life cycle. The system should be tested against functional specifications produced during the Requirements Definition and Analysis stage. For this reason, adequate specifications for quality must be included and the requirements stated in a way that can be tested. During the Design stage, when components of the system are first identified, the system test plan is produced as part of the system specification. The system test plan defines how modules or programs of the system are to be sequenced and pieced together. It defines the order of integration, the functional capability of each version of the system, and the responsibilities for producing code that simulates the functions of nonexistent components. Development of test cases for system testing is accomplished during any or all the Design, Programming, and/or Validation stages. System test cases consist of test data and scenarios supplied by the user, programmer, and/or system testing component.

2.5 Operation Stage

The Operation stage begins after a software product has been validated to the satisfaction of the user and has been accepted for production processing. This applies to both new development efforts and enhancements to existing systems. The operation stage includes the executon of software and the interface between development and operational components necessary to ensure proper execution.

2.6 Review Stage

The Review stage begins after software is operational and basically in a maintenance mode. The purpose of the review stage is to provide for periodic performance evaluations of operational software systems. The evaluation must address performance characteristics from the perspective of the user, development, and operations components with a view toward alleviating errors and recommending improvements. Review can be initiated based upon management direction, problems, software age, or scientific selection. Scientific selection is the preferable method since it is the least subjective. The review itself is most effective as an internal peer group activity conducted at the project level. Activities associated with this stage include-

- qualitative and quantitative data gathering;
- data evaluation; and
- trend analysis.

3. Overview of the Five Elements

Establishment of a SET in any ADP environment consists of the development of a detailed plan that describes and integrates the five major elements of-

- standards;
- procedures;
- tools;
- quality assurance; and
- training.

This combination of standards and guidelines, procedures, tools, quality assurance and training forms the basis for a software technology [3].

3.1 Standards

The term "standard" can be defined as that which is established by authority, custom, or general consent as a basis for comparison. Data processing standards can be grouped into two major categories: methodology standards and performance standards.

Methodology standards are uniform practices. Performance standards are metrics to evaluate performance. Methodology standards are rules of procedures or instructions on "how-to-do-it." Performance standards specify how well a function should be be performed by people, software, and machines. For example, programming methodology standards would indicate how a program is to be coded (i.e., what coding form to use, how characters should be hand-written for input, the format of the source code listing, etc.). The programming performance standards would state how long program coding should take, given the experience of the programmer, complexity of the job, and other relevant factors or constraints.

3.2 Procedures

Procedures are methods of doing business within an organization. They define processes which are followed on a project or the manner of proceeding with a task. For a SET, procedures are really the "what-to-do" and "when" in performing software tasks.

There are several types of procedures particular to a SET. There are undocumented and documented procedures, as well as manual and automated procedures. Undocumented procedures are the methods or processes which are performed based upon tradition. Normally these procedures are communicated from person to person by word-of-mouth. Approval chains or signoff procedures are quite often examples of this type of procedure.

Documented procedures are formal written procedures in a data processing shop such as software run instructions, test case development steps, software testing process, etc. They serve as a source of reference to personnel for maintaining consistency and uniformity in completing software development tasks. Documented procedures eliminate much of the trial-and-error in determining "what-to-do" and "when-to-do-it." Also, documented procedures are an invaluable source of training material when integrating new personnel into the organization. All procedures used in the software life cycle should be documented including those that were normally performed only by tradition. If not documented, the procedures will never become public knowledge or practice, and will not be adopted and accepted by the data processing personnel.

Manual procedures involve activities such as: desk checking code, obtaining management approval for proposed system enhancements, and interfacing with users to identify system requirements. Automated procedures involve the use of software tools, such as: language analyzers, code optimizers, requirement analyzers, and programming support tools.

3.3 Tools

"A software tool is a computer program that can automate some of the labor involved in the management, design, coding, testing, inspection, or maintanance of other programs [4]."

There are many types of tools in widespread use throughout the ADP industry. Their use has become almost essential to the effective use of computers for any application. These tools range in size and complexity from simple aids for individual programmers to complex tools that can support many software projects at the same time.

Tools are important because they can be used to produce software faster, more accurately, and more uniformly, while significantly improving personnel productivity. As a result, their use can become an important part of software development. More importantly, tools represent a class of software that can be used and reused within many different environments. Thus, the use of tools provides the opportunity to reduce costs and improve productivity while decreasing development time.

3.4 Quality Assurance

Quality Assurance (QA) is the formal process of measuring or evaluating, the degree to which software meets standards (e.g., alignment of code or percentage of logic executed during a test), and/or prescribed requirements (e.g., in the areas of accuracy, reliability, etc.). The primary purpose of QA is to develop and maintain better software products than those which would otherwise have been developed and maintained using traditional methodologies without measurements or controls.

3.5 Training

The purpose of training is to create, through some type of learning experience, a permanent change in a person's behavior so that the individual reliably performs in a certain prescribed manner. The types and quantity of training required for a SET depend heavily upon an organization's personnel background and the make-up of the SET itself.

Webster defines training as "a process or method to lead or direct growth; to form by instruction, discipline, or drill" [5]. Training implies a method (procedure), change (behavior modification), and result (growth or performance). Training in terms of the SET requires the identification of target training areas and specific plans of action to bring personnel to a higher level of knowledge and performance. Since the establishment, institutionalization, and implementation of SET is evolutionary, the training plan should be structured to facilitate this process and provide training which closely corresponds to the SET.

4. Planning the Development of a SET

4.1 SET Planning

Developing a comprehensive SET requires detailed planning, organizing, personnel coordination and, most importantly, management involvement and support. If the SET development effort is visible in an organization, the chances that a formal SET will be implemented increase substantially. Establishing such an effort as a planned program in an organization's plan is the first step in ensuring that the process will receive the necessary attention. Involvement by all parts of the organization in the development of a SET will increase the potential for successful implementation.

Two approaches in planning the development of a SET can be used - a top down approach and a bottom up approach. A top down approach develops a theoretical structure of software engineering, and expands this to successive levels of detail until all tasks are completed. The bottom up approach identifies the current engineering process, and from this identifies which elements of the technology are missing. An evolutionary process of substituting tools for existing processes, developing missing procedures, standards, etc., by trial, analysis and modification, will evolve the current system towards the desired SET.

It is important that the SET development plan provide a basic framework, direction, and overall schedule for the project. The plan should be structured so that it can be updated on a periodic basis and should address all five elements of-

- standards;
- procedures;
- tools;
- quality assurance; and
- training.

An organizational planning process involves both tactical (short term) and strategic (long term) objectives. These objectives are usually identified in an organizational plan as a ranked group of milestones. The ranking process is conducted by management whereby each milestone is evaluated using criteria which mirror the goals of an organization. Top level management usually establish these criteria which are then used to prioritize and rank organizational objectives. The establishment of a SET requires detail knowledge of an organizational plan to establish standards, buy software tools, develop procedures, and plan an adequate computing facility which will best support organizational objectives. The SET Development Plan must strike a balance between supporting organizational objectives and how rapidly the engineering technology is implemented. As part of the plan, an analysis of the current engineering process should be conducted and should be organized so that it can easily be compared to the new, planned process. This comparison will help identify actions needed to upgrade the process. The analysis should also identify high pay-off areas as this will help establish priorities. From this analysis will emerge the long range strategy and a detailed plan [3]. Figure 2 shows a possible outline for a SET Development Plan.

4.2 Model for Developing a SET

The method used for establishing a SET depends on a particular organization and its staff. There are, however, some principles which are characteristic of any model for developing a SET. A SET should be flexible in order to maintain an up-to-date technology. There should be periodic checkpoints to enable personnel to be brought in tune with the established technology. It should allow room for experimentation with modern software practices. Finally, it should allow integration of software tools to automate manual software activities where possible.

It is good practice to establish periodic baselines during the SET development process because developing a technology is generally disruptive to an organization. It frequently changes the way people produce and maintain software. It requires training and makes old procedures obsolete while requiring that new ones be adopted. To minimize this disruption, there should be planned periods during the SET development process where technology is stabilized so that personnel can upgrade their skills and put into use the software engineering principles formulated thus far.

Figure 3 presents a simplified model for developing a SET. The target SET developed from application of the model must exist within the boundaries of a particular environment. Environment represents a multi-dimensional concept that identifies the many components of an organization and their relationships. Some important attributes of an ADP environment include-

- size of the software organization & personnel mix;
- type of organization (private, government, etc.);
- applications (scientific, MIS) and language(s);
- development environment (batch, interactive);
- program running environment (batch, real-time, etc.);
- computer type; and
- involvement in tools develoment [4].

These attributes should be evaluated and quantified in order to compile an organizational profile. This profile, coupled with key SET elements will serve to establish a custom fitted SET baseline. Establishing a baseline SET will require a proper environmental mix of SET elements within the software life cycle stages that are balanced against an organization's needs and requirements.

4.3 Identifying Standards, Procedures and Quality Assurance

Identifying standards, procedures and quality assurance for a SET is a large and sometimes confusing task. First there are questions such as what software practices should be standardized and, what processes should be developed? Each organization developing a SET must eventually answer these questions. Although answers to the questions will be different, the items to be considered can be grouped into categories. Only then can the translation of these categories into preliminary, and ultimately, fully documented standards, procedures and quality assurance methods that fit into an organization's environment be performed.

4.4 Standards Considerations

One important area of consideration regarding standards is that conformance with Federal information Processing Standards (FIPS) is a regulation for all Federal agencies. However it should be noted that there are waiver procedures for most of the FIPS. The FIPS Publication Series of the National Bureau of Standards, U.S. Department of Commerce [6], is the official publication relating to standards adopted and promulgated under the provisions of Public Law 89-306 (Brooks Act) and under Part 6 of Title 15, Code of Federal Regulations. These publications consist of guidelines and mandatory standards for the utilization and management of computers and automatic data processing systems used in the Federal Government.

4.5 Integration of Software Tools

The introduction of a tools oriented SET into a

1. PROJECT INITIATION

1.1 Purpose of SET Project
1.2 Objective and Goals of SET
1.3 SET Development Methodology
1.4 Organizational Responsibilities
1.5 Resource Estimate, Schedules and Milestones

2. SET REQUIREMENTS

1.1 Task and Deliverables
1.2 Phased Milestones
1.3 Applicable Standards
1.4 Manual and Computer Procedures
1.5 Quality Assurance Mechanisms
1.6 Tool Functions
1.7 Environment Considerations

3. CURRENT ENGINEERING PROCESS (Similar Structure to (2))

4. PRELIMINARY SET DEFINITION

4.1 Preliminary Tool Configuration
4.2 Preliminary List of standards
4.3 Preliminary Procedures
4.4 Preliminary Quality Assurance Mechanisms
4.5 Training Plan

5. SET DEVELOPMENT PLAN

5.1 Pilot Projects
5.2 Tool Selection and Acquisition
5.3 Standards Development
5.4 Procedures Development
5.5 Quality Assurance Establishment
5.6 Training
5.7 SET Measurement and Evaluation

Figure 2: Plan Outline for Developing a SET

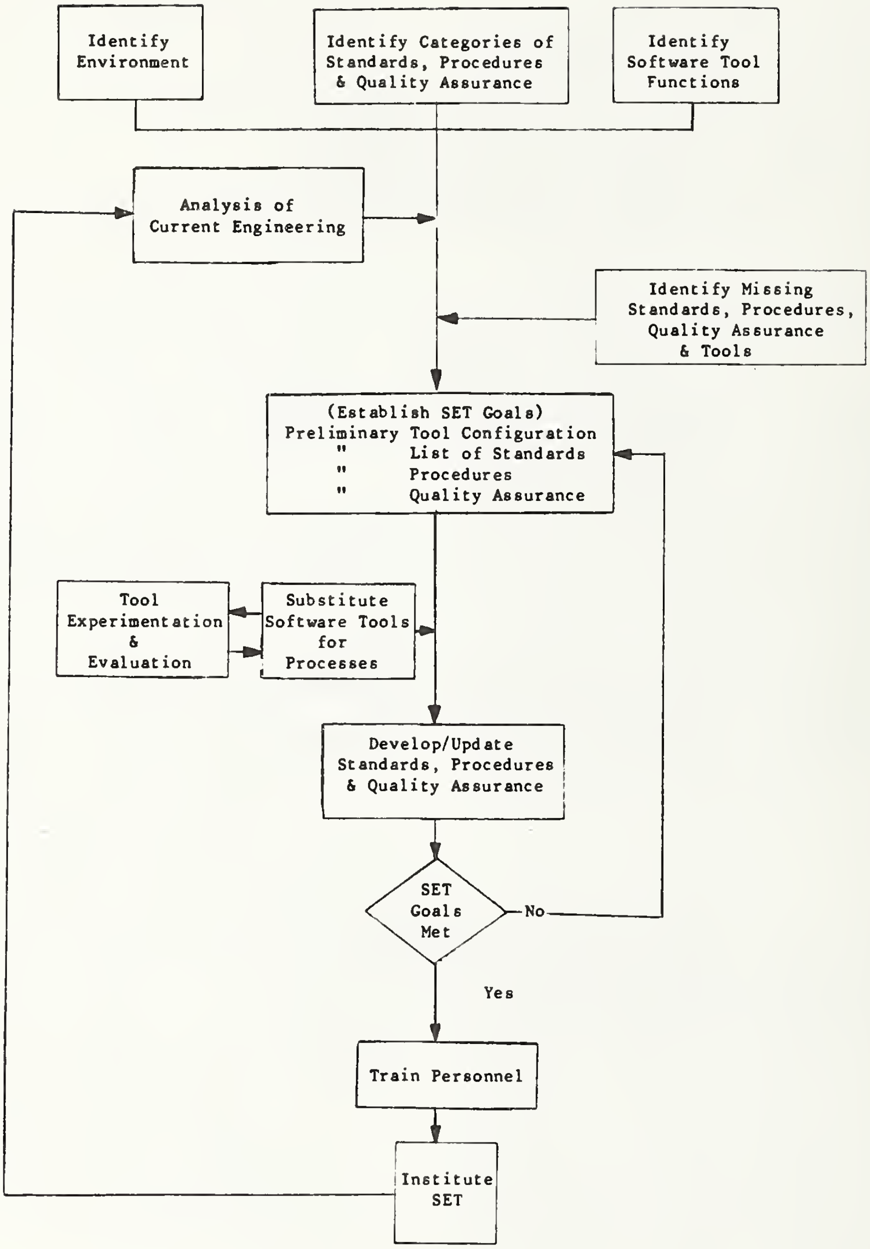

Identify Environment
Identify Categories of Standards, Procedures & Quality Assurance
Identify Software Tool Functions
Analysis of Current Engineering
Identify Missing Standards, Procedures, Quality Assurance & Tools
(Establish SET Goals)
Preliminary Tool Configuration
" List of Standards
" Procedures
" Quality Assurance
Tool Experimentation & Evaluation
Substitute Software Tools for Processes
Develop/Update Standards, Procedures & Quality Assurance
SET Goals Met
No
Yes
Train Personnel
Institute SET

Figure 3: Model for Developing a SET

software development environment will impact the organization in several ways. To cope with these changes organizations will need to-

- institute a methodology for software development based on life cycle;
- establish and enforce organizational standards for software development;
- supply automated tools to facilitate the software development process;
- supply training for personnel in modern programming practices; and
- provide the management commitment to develop and sustain the environment.

The recommended approach to introducing a tools oriented SET into a software development environment is to solicit an abundance of user involvement; proceed to successively more advanced levels of the SET in a systematic, coordinated manner; constantly obtain and evaluate feedback; and if necessary, make continual changes to the SET to ensure its appropriateness to a particular environment. A properly integrated tools oriented SET will yield more maintainable and error-free software more productive programmers, improved software management, and dramatically increased control over the software life cycle process.

5. Phasing Into the Technology

5.1 Tool Identification

SET is an evolutionary concept that migrates from a simple technology, through several higher levels to an advanced (state-of-the-art) technology. SET technology differentiates from existing software development methodologies in that it relies heavily on the use of tools. Tools are used to accelerate the development process and to take advantage of extensive simulation and modeling techniques which facilitate high quality software development. Examples of these include-

- simulation tools;
- development tools;
- test and evaluation tools;
- operations and maintenance tools;
- performance measurement tools; and
- programming support tools.

Early software development environments consisted of a compiler and a linking loader. Later environments included text editors and debuggers, informal requirements and design methods, and simple programming standards. Many new methods and software tools have been formulated and built during the last decade.

A software tool environmental review requires an analysis of products generated during a software development project and the methods and tools useful in generating those products. The software life cycle model emphasized the importance of both intermediate and final products. Each unique software life cycle model has classes of products associated with particular life cycle phases while others transcend them. Requirements definition products serve as a means of communication with the customer. They are often informal and may simply be notes tacked up on office walls, compiled in a cut-and-paste mode. They may also be expressed more formally in a language like Structure Analysis and Design Technique (SADT) [7]. Requirements specifications are formal contractual documents that define the system to be built. They are constructed after requirements definition and, ideally, are represented in a formal language on graphical notation such as Problem Statement Language (PSL)/Problem Statement Analyzer (PSA) [8]. A test plan based on the functional properties of the system described in the requirements specification is also an important product. The plan should define both test data and expected results as well as procedures for running the tests [9].

The design of a system is often divided into preliminary (or architectural) design and detailed design. Both phases result in design products that may be either informal prose descriptions or diagrams that are generated using a formal design methodology such as Structured Design [10]. In addition to system documents, the preliminary design may also result in the generation of "build plans." Build plans describe the order in which modules or parts of the system are to be designed and built. The preliminary design may also be used to construct schedules, budgets, resource management procedures, milestone charts, and maintenance procedures [11].

In addition to the source and object code modules, the programming stage involves the construction or modification of management products (budgets, schedules, etc.), user manuals, discrepancy reports, and code-based test plans. The programming stage also includes the generation of reports of all testing and code analysis activities. Validation products ensure that the software system meets system functional specifications and requirements, and also that as many as possible of the "Defects" in implementation are removed. This provides strong indications if the system is constructed according to the design document and meets all appropriate criteria as identified in the requirements/specification document. Tangible products will appear in the form of exception reports and/or warnings about anticipated problems. Test and evaluation tools and techniques can be used to support the validation process.

Systems operation and maintenance are concurrent activities. These are products important to and either used or generated during maintenance. They include configuration specifications, change control procedures and plans, regression test data and results, cross-reference documents, and all requirements and design specifications. The review stage of the software life-cycle is an ongoing activity within each stage. It is highlighted at key milestone points where products from one stage enter into another. Therefore the products generated in each life-cycle stage constitute the review products.

A tools oriented SET is an evolutionary concept that allows migration to successively higher levels of SET by implementing more complex tools and their related standards and procedures. This migration process moving into an ever increasing tool complexity environment will be both time consuming and difficult and at a minimum will require extensive planning and coordination. Obtaining top level management commitment is a vital step in the process.

5.2 Tool Integration

A common argument against tool integration is that it imposes an inflexible development methodology on the development staff. One imagines a complex tool. To use Tool A, you must first use Tool B, and the output from A is always processed by C and then D unless there is input from E. The idea that an integrated tool environment must consist of a complex structure of interconnected tools is misleading. The fundamental feature of some of the best known software support systems is not their interconnections, but the use of common kind of data objects by all tools or facilities. The properties of the basic data objects and the knowledge that different parts of the system have of objects of this kind characterize the degree and type of integration within the system.

A software engineering database can be used to build an integrated software development environment. The database provides an integrating and unifying medium for interfacing tools without forcing them into a complex structure of interrelationships. Tools obtain their information from the database and return their results to it without having to interface directly with other tools.

An integrated tool system which uses a common database eliminates the need for multiple copies of the same information. The existence of different copies of the same information for different tools often creates a consistency and synchronization problem. Every time one version of a collection of information is updated or changed, other copies must be changed. Both the expense and tedium of this virtualy rules out the practicality of using certain collections of tools unless they can be modified to work off the database. Tools which satisfy the requirements for tool compatibility (parameterization of input/output locations and the ability to call one another) can be attached to a software engineering database with interface for "data translator" routines. In order to maintain flexibility, it is important to avoid building bridges between pairs of tools. The bridges should instead be built between the tools and the database [11].

5.3 Benefits of SET Baseline

Entry into a SET is usually indicated by establishing a SET baseline. This is the minimum set of tools and technology needed for an initial SET. It is the starting point from which to migrate to a more advanced SET technology. These requirements represent the critical mass of a SET. They also translate into a series of benefits which have common applicability to any installation. Therefore, an initial baseline SET would-

- increase productivity;
- upgrade skill capabilities;
- automate routine aspects of software design; and
- reduce the time and cost of software maintenance

For example, software tools meeting baseline requirements could provide an increase in the quality and quantity of software code, be much simpler to use than writing new programs, have expanded software design capabilities, and be self documenting. These tools together with the standards and procedures to use them could constitute an initial baseline SET.

Each organization should evaluate its own position in relation to a SET. Any environment that provides tools that supply at a minimum these stated benefits, can be said to have established an initial baseline SET.

Once the baseline technology is in place, and personnel have been trained in its use, the next technology "release" should be planned. Modifying a SET requires a certain amount of overhead, such as the retraining of personnel in the usage of new or enhanced features. For this reason, periodic versions (or releases) of the SET should be planned and implemented, incorporating more advanced tools, modifications and/or enhancements to the existing standards and procedures, etc. The SET documentation should be appropriately updated, and personnel trained in the use of the new enhanced features.

5.4 Establishing High Pay-Off Baselines First

The net effect of establishing a SET baseline technology is to stabilize an ADP organization. Upon reaching a SET baseline, an organization will be in a much better position both managerially and technically to proceed toward the various higher levels of a tools oriented SET.

The use of software tools offers immediate benefits in terms of time and cost savings. The integration of software tools and advanced management techniques, such as structured walkthroughs, chief programmer teams, and program reviews offer high pay-off potential for SET. Some software development techniques which have consistently been found to offer high pay-off potential when used with a tool oriented SET include-

- requirements analysis and validation;
- baselining on requirements specification;
- complete preliminary design; and
- process design [12].

Installations to maximize their payback from investment on new development should concentrate on software tools and technologies which apply to the requirements and design stages of software life-cycle management. Mature organizations wishing to maximize their investment return on existing software should concentrate on programming and software validation tools.

5.5 Training in Relation to SET

The types and quantity of training required for a SET depend heavily upon an organization's personnel background and the make-up of the SET. In most installations, data processing personnel are not trained in the engineering disciplines. Therefore, a significant upgrade in skills will be necessary so that individuals can take full advantage of the new technology.

Any training plan should utilize a balanced approach to provide adequate training to develop all skills required at a particular job level and category within a SET.

The identification of tools, quality assurance, standards and procedures to be used at critical points in the plan is important. Tool and technique training properly integrated into the training program will ensure that an organization is in tune with state-of-the-art technology.

A SET will progressively lead an ADP organization toward a state-of-the-art operation. Employees will be exposed to a rapidly changing, complex environment. Everyone connected with the SET will participate in the changes and feel its affects which, without proper training could become highly stressful. The mitigation of stress during this dynamic period should be a major objective of the SET training.

5.6 Maintaining an Up-To-Date Technology

To maintain a continuing level of technology that will keep pace with a rapidly moving software industry, an organization should consider either expanding its current information procedures and facilities or instituting new ones to ensure its personnel and technology stay up-to-date. Most data processing organizations maintain some type of library. It may be formal as in the case of a central library or as informal as books on a table in a conference room. The purpose of the library is to provide a ready source of reference on information relating to data processing. Just as a data dictionary is vital in the management and use of a DBMS, a current library is vital as a rich source of information on system's development and management.

Exposure to modern concepts in tools and technology can help to increase individual productivity. Concepts and disciplines being taught in contemporary academic institutions are quite different than those taught in prior years. Entirely new disciplines, such as Computer Aided Design (CAD) and Computer Aided Manufacturing (CAM), with emphasis on quantitative evaluation and return on investment require much more than knowledge of theory [13]. Many data processing professionals have either limited exposure to these concepts or no formal exposure. SET requires an organization to re-tool its environment with software tools and technology. In conjunction with this, SET will require individuals to re-tool (train) using quantitative skills to properly take full advantage of the new tool technology.

Success in implementing a SET is a direct result of a balance achieved between software life cycle management and the application of a software tools technology. The balance is delicate and is maintained informaly by using the standards and procedures review process and formally through periodic project meetings with upper level management. It is at these meetings where SET performance is measured against the project plan, future milestones identified, and specific direction is provided from upper level management to ensure that the SET technology is kept on track. These top level management meetings are vital in the SET project management process. They provide performance information to upper level management on SET project performance and provide the opportunity for lower levels of ADP management to receive specific feedback on the SET project as well as any new information which could impact or enhance the SET technology.

6. Description of a SET to be Developed

6.1 SET Composition

A SET is composed of standards and guidelines, procedures, quality assurance, tools, and training applied to software life-cycle stages in a planned balance within a carefully engineered environment. The technology reflects the process of replacing manual with automated functions within a software development environment. The development process is characterized by the identification of baseline technologies which incorporate software tools and techniques with succeeding baselines equating to higher levels of software quality, performance and productivity. Baselines are custom engineered for a particular organization. They identify criteria to measure performance and specify the required tools and technology needed to reach a target performance plateau specified by and for an organization. An organization adopting a SET enters into a constant review and evaluation process to ensure their conformance with existing SET baseline requirements and to identify the tools and technology for higher baseline SET levels.

The recommended SET development plan for a software engineering technology should include the following major phases:

- Existing engineering analysis,
- SET baseline planning (goals),
- SET baseline development/upgrade.

The exact nature and mix of software tools and techniques must be determined during the initial baseline identification process. Areas of software technology having the highest pay-off should be given prominent consideration in an initial baseline SET development plan. The composition of any technology is shaped by factors within the particular organization, such as its management structure for software development, its staff, its physical workspace and computer environment, and its applications. As a result, different organizations may have different software engineering technologies.

The initial baseline SET should concentrate on the programmatic software environment. After the initial programmatic oriented SET has been established, the technology should be expanded to encompass other application areas. Subsequent SET baselines should include more advanced tools and techniques.

6.2 General Workflow for SET Development

The workflow for SET development is a dynamic process subject to many factors having environmental and technical impact. A properly engineered and integrated SET Matrix translates into a level of a SET. Replacing manual functions in the matix with software tools and techniques and institutionalizing their use changes the complexion of the matrix and positively impacts the technology. Technology levels identify higher plateaus of software engineering in the form of organizational goals. They identify and define target functional areas which can benefit from a carefully engineered SET plan. The plan should define the steps necessary to upgrade the SET by the application of software tools and technology. A technology plan is composed of a series of functional upgrade plans. Each particular plan does not in itself represent a technology level but rather a small portion of one. The combination of individual functional upgrade plans targeted to meet specific organizational goals at a planned level of productivity and performance constitute a baseline SET. The completion and institutionalization of all plans within a given technology level identifies that an organization has reached that baseline level for SET.

6.3 Subdivision of the SET for Work Assignment

SET implementation requires detail understanding of the technology and its organizational considerations and impact. A reflection of this understanding is the development of a plan which identifies software life-cycle stages and is broken into distinct phases.

"Establishing standards, identifying and acquiring the appropriate computer resources needed, introducing a few mature tools and progressively building on these capabilities is a good approach. Prioritizing the steps and identifying and acquiring high payoff tools and techniques need to be clearly mapped out. This should be done at the outset and reviewed along the way" [14].

Prior to entering a tools oriented SET, an organization should plan initially to phase into a SET by establishing a baseline technology. The advantage of this would include-

- stabilizing an organization;
- establishment of a pilot project;
- phased tool implementation;
- identification of high payoff areas;
- testing of SET on limited scale; and
- slow exposure to software tools.

The formalization and eventual institutionalization of a SET is an extremely complex and delicate process. Adding to the complexity is the fact that the SET undergoes continual evaluation with identified changes being incorporated back into the technology. A good first step in establishing a SET is to experiment with the tools and technology. This experimentation should be conducted on a small scale utilizing several types of software tools. This testing may identify difficulties and problems that previously may have gone undetected. The realization that these exist may cause a re-evluation of the SET Development Plan. The intent here is to advise slow progression into a SET environment. Changes in scope and content may be required to properly tailor the SET plan to the environment. Some of the project milestones scheduled to be included in the baseline SET may better be handled at higher baseline SET levels. This determination can best be performed after an experimental stage where organizational SET limitations can be identified and factored into a strategic organizational SET Development Plan.

7. Summary

Computer software is expensive to develop and it is generally not well contolled. This is largely due to the lack of discipline which accompanys the development and maintenance activities associated with software. Software engineering is the application of a discipline and is no more than adaptation of the principles found in other engineering fields, to computer software so that the capabilities of computer equipment can be made useful to man (i. e., not costly and can be controlled). Establishing a SET within an organization is not a trivial effort for it requires involvement by all segments of the ADP organization and it changes the way people have been accustom to developing software. To be encompassing it must consider all stages of the software life cycle from requirements definition and analysis through operations and review. Further, it must apply the SET elements of standards and guidelines, procedures, tools, quality assurance and training to these life cycle stages. Also, there needs to be a phased plan for establishing SET baselines in order to stablize the people and the organization to minimize disruption of present software development activities. Last,and most important, there must be liberal use of software tools to automate the disciplines established and to reduce the amount of labor associated with software development and maintenance.

References

1. Boehm, Barry W., Software Engineering - As It Is, 4th International Conference on Software Engineering, IEEE Catalog No. 79Ch1479-5/C, 1979.

2. Pressman, Roger S., Software Engineering: A Practitioner's Approach, McGraw-Hill Book Company, 1982.

3. A Software Tools Project: A Means of Capturing Technology and Improving Engineering, Report OSD-82-101, Office of Software Development, GSA, February 1982.

4. SOHAR, Inc., Guidelines for the Introduction of Software Tools into a Programming Environment, Contract NB79SBCA0273, Task #3 National Bureau of Standards, April 1981

5. Webster's New Collegiate Dictionary, G. & C. Merriam Company, 1979.

6. Federal Information Processing Standards Publications (FIPS PUBS), NBS Publications List 58, U. S. Department of Commerce National Bureau of Standards, February 1982.

7. SADT, The Softech Approach to System Development, Software Technology Co., Waltham, Mass., January 1976

8. Teichroew, D., and Hershey, E. A., PSL/PSA: A Computer-aided Technique for Structured Documentation and Analysis of Information Systems, IEEE Trans. Software Eng. SE3,2, 1977

9. Panzl, D.J., Automatic Software Test Drivers, Computer 11,·1978

10. Yourdon, E. and Constine C., Structure Design, Printice Hall, Englewood Cliffs, NJ, 1979

11. Howden, William E., Contemporary Software Development Environments, Communications/ACM, May 1982

12. Rochkind, M.J., The Source Code Control System, IEEE Trans. Software Eng. SE-1, 4, 1975

13. CASA, CAD/CAM, Society of Manufacturing Engineers, Library of Congress Catalog Card Number 80-69006, 1980

14. Krygiel, Annette J., Lessons Learned on the Road to a Modern Programming Environment, Defense Mapping Agency Directorate for Systems and Techniques, 1980.

Characteristics of Software Development Team Structures and their Impact on Software Development

Anneliese von Mayrhauser

Illinois Institute of Technology
Department of Computer Science
Chicago, IL 60616

Several of the team structures proposed in the literature such as Chief Programmer Team, Surgical Team, Revised Chief Programmer Team advocate a separation of tasks for a programming team resulting in specific roles for the members on the team. An analysis of these roles with respect to personality and task requirements is presented which enables a better tailoring of these team concepts to specific projects with a given staff. Based on the definitions of the various roles of the different team structures requirements for a particular position are derived and suggestions are made how to select the most appropriate team structure for different types of projects and known people characteristics. Depending on how well team structure and its requirements match problem and people characteristics indicators can be derived pointing out possible problem areas before they occur so that corrective action can be taken before schedules and/or budgets are overrun and team members become dissatisfied.

I. Introduction

Almost since the beginning of programming the problems programmers were working on were big and complex enough that they had to organize and work in groups. In the beginning these groups were highly ad hoc and unstructured which in many cases resulted in a lot of entropy and low quality but costly software. The programming team as a more structured form of organization was invented as a cure for many a chaos oldtimers call war stories. Pretty soon people realized that just as a program needs not just any, but a specific kind of structure, human interaction between programmers working on the same problem needs structure, too. One of the first team concepts which was created was that of the Chief Programmer Team (CPT) /BAKE72/, /MILL83/. Many of the high expectations this concept generated were soon shattered, however, because the problems with software development did not go away. Other team concepts were born, most notably among them the Surgical Team (ST) /BROO75/, the Revised Chief Programmer Team (RCPT) /MCCL81/, and Egoless Programming (EP) /WEIN71/. The problems, i.e. cost and schedule overruns, unreliable software, high turnover still persisted, in spite of these new organizational forms and in spite of structured design, structured programming, structured testing, structured walkthroughs - one might be tempted to say structured anything.

People started investigating the nature of these structures, for instance how team structure, most notably reporting structure, influences program structure and they realized that indeed it does to a great extent. On the other hand we know that most problems have an indigenous structure, e.g. passes of a compiler or the hierarchical structure of many application programs. The question then becomes whether the team structure and the structure of the problem are compatible. If they are not, difficulties are bound to arise, because following the team structure then goes against the grain of the the problem structure and vice versa.

It is important to realize that the need for structure in the product stems from the desire to reduce the complexity of the problem and its solution by partitioning it into logically self-contained parts having the simplest interfaces possible. Likewise the motivation underlying the definition of programming team concepts arises from the need for dividing tasks into self-contained parts which can then be assigned to members of the team and carried out by them with the least need for further clarification and even communication with other team members. Another motivation was specialization. This is rooted in the belief that a higher degree of specialization is more economical, because most people are not equally talented or skilled in all tasks during the software development process, and if we can assign the most highly skilled person to a task we get better quality and higher efficiency. This is a concept which sometimes has been employed to advantage in the software development industry as it has in other industries since Smith in 1776 first advocated the concept of division of labor /SMIT76/ which was later taken up and further developed by C. Babbage in 1832 who emphasized decreased learning time and increased skill due to repetition as some of the advantages of specialization /BABB32/. Later "scientific management" /TAYL11/ went even further in specialization leading to the attitude underlying assembly line work. In programming we also have the possibility of considerable specialization and standardization concepts which, taken to the extreme, can lead to assembly line programming with all the pros and cons assembly line work is known to have.

Through research in ergonomics and psychology we have learnt that different degrees of specialization require different personality characteristics. For example, if a very strictly specified and standardized task is assigned to an individual with high creativity needs and needs for high growth and responsibility, this person will very likely feel bored and dissatisfied because these needs are not being met. Considerations like these lead to another set of factors which have to be considered: personality traits which either facilitate or impede a set of task assignments within a chosen team structure. Lastly the team has to fit into the overall organizational structure, otherwise too much external friction or too little involvement with the rest of the organization may result.

The remainder of this paper will review the most commonly advocated team structures, i.e. CPT, RCPT, ST, and EP, and analyze them with respect to personality requirements and degree of specialization involved. It will investigate the types of projects which are most suited for which team concept and present a set of guidelines how to pick the right environment for the right task.

II. Review of Concepts in Task Analysis and Design

As pointed out earlier, the advent of programming team structures was motivated by the need for specialization. One improvement hoped for was a decrease in the overhead effort due to the need for communication between all people who have to interface with each other. A second was that we would no longer need the "renaissance-man (or woman) of programming" who had to be able to do all jobs involved in software development equally well. Since people often have very specific specialized talents, this notion had been rather unrealistic to begin with. Programming is a complex task involving a multiplicity of functions and at times as we know from software for space flight to software for automatic shutdowns of nuclear reactors may be rather critical with a high need for reliability.

Before selecting a specific team structure for a software development effort we have to investigate the following dimensions for an adequate fit:
1. Degree of formalization to enable proper management control for quality, schedule, cost, etc. according to goals and priorities.
2. Interaction with environment. A development team develops software for a user and thus has to relate to the user or its representative. Furthermore the team is also part of the organization within which it is placed.
3. Product structure. This refers to the final software product as well as the products of intermediate stages (requirements, specifications, design, etc.)
4. Needs of the individual and requirements for the individual. Here we include the degree of clarity of task assignment, the amount of communication needed as well as an assessment of adequate task attributes for the

individual's motivational needs. We need to look at what is called task analysis and design. One of the theories which is helpful in this context is the Job Characteristics Theory by Hackman and Oldham /HACK76/, /HACK80/. This theory states that there are 5 different core job dimensions which influence three major psychological states, the primary determinants of motivation. The 5 core job dimensions are:

* skill variety - how many different skills does a job require?
* task identity - to which degree is a job done from beginning to end with a visible outcome
* task significance - what impact does the job have on the environment
* autonomy - freedom and independence to do the work (schedule and procedures)
* feedback - how much information about quality of work performance is given during the work

The 3 major psychological states are:

* Experienced meaningfulness of the work
* Experienced responsibility for work outcomes
* Knowledge of results

Obviously not everybody needs the same amount of task significance, autonomy, feedback, task identity or skill variety. How much an employee needs in these core areas in order to experience that he/she is doing a meaningful job providing enough responsibility and knowing enough about the results to be satisfied depends on that person's growth need. Some individuals require high scope tasks whith very high levels in all job dimensions mentioned, others become overly stressed when a task requires too many skills, is too loosely defined (too much autonomy) or does not provide instant feedback at the end. Beyond job characteristics we consequently have to look at the skill level of a person (the more skills the less stressful a situation, but an overly skilled person may get bored) but also at the three basic work motives:

* task - performing it and becoming more skillful
* relationships - popularity, interaction
* influence and direction over people

Motivation can be positive or negative and vary in intensity. We have mentioned task oriented motives, external stimuli. Internal motivators are personality traits. It has been said that DP professionals have a very high motivating potential due to their high growth need (/COUG78/) which is connected to the work motives task and influence and direction, but that their social needs (relationships) are low. This is very similar to the motivational profile for engineers (/STEV77/). This usually also means that their skills in this area are not as well developed and/or that they experience a higher stress level when faced with such tasks than people with higher relationship needs. Stevens and Krochmal represent that these personality traits result in the following "turn-ons" and "turn-offs" for the individual (/STEV77/, p.168):

Turn-ons:
1. Moving forward on project when he/she feels it is appropriate, having and maintaining control.
2. Being able to measure own progress.
3. Having to keep in touch only with project progress (that affects him/her).
4. Brief, to the point, pragmatic communication
5. Practical work
6. Personal hoals, in specific project goals.

Turn-offs:
1. Waiting for politics, etc. or things he/she cannot control.
2. Not knowing how he/she is doing or how work is progressing
3. Having to keep up with things that don't directly concern him/her, e.g. administrative meetings.
4. Policy statements, personnel forms, regulations.
5. Having to remember feelings, birthdays and social events.
6. Group concerns and organizational goals.

Although some of these traits shed light on current problems in software project management, notably staffing and control (/THAY82/), and explain the preoccupation with tools and development methodologies versus proper management procedures (/ZOLN82/), this list of motivators and demotivators should be regarded as a checklist for traits to be considered and to evaluate rather than assuming that they are always present to a high degree. After all individuals do differ. If we want to select the proper team structure for a software development effort we have to look at these factors to define tasks with the proper attributes for the members involved so that high internal work motivation, high quality work performance, high work satisfaction and low absenteeism and turnover is facilitated as much as possible. With these thoughts in mind let us now turn to a review of the individual team structures.

III. The Chief Programmer Team (CPT)

The CPT provides a high degree of formalization within a strict organizational structure, clear leadership through the Chief and the possibility for specialization through functional separation. The reporting structure within the team is explicitly defined, as are the relationships between team members. Some functions are explicitly defined. The rigid structure facilitates management control, visibility of product and personel, communication and product structure. It also tries to guarantee continuity by ensuring that at least two team members, the Chief and the Backup programmer are familiar with every aspect of the project. The nucleus of the CPT involves the following individuals:

1. Two technical experts, the Chief Programmer and the Backup Programmer. While the Chief Programmer is the undisputed technical leader who is responsible for the team's success, who develops all documents of the early phases (requirements, specifications, design), who codes and tests the critical parts of the system and (closely) guides and supervises the other team members, the Backup Programmer although he does not have the decision making power of the Chief Programer, acts as a backup leader and peer to him/her. As such he has to be totally familiar with the project in order to be able to take over leadership when necessary and to participate in all important technical decisions. He is responsible for the test plan and also usually does research work for the Chief Programmer. Obviously these two functions require considerable expertise in software development, the Chief Programmer also has to have sound management experience. The other functions do not require technical and managerial expertise at this high a degree.
2. The clerical assistant or programming secretary makes sure that the documents are current and visible and maintains libraries, test data, test results and project documentation.
3. The programmers are junior personel who implement the code according to the Chief Programmer's directives. Optionally there may be a project administrator who reports to the Chief Programmer and takes over some of the administrative tasks.

When we look at the four team dimensions, we can see that this team concept rates very high in terms of dimension 1, degree of formalization. Provided that the Chief Programmer does his/her job correctly there is adequate possibility for management control for quality, schedule, cost according to priorities and goals the Chief Programmer sets for the team. No conflicts arise due to ambiguous reporting structure, or conflicting goals set by several people. The team is relying for interaction with its environment wholly on the chief programmer. His responsibility is to talk to the users as well as to represent the team to the rest of the organization. Since the team structure is hierarchical the product structure tends to be as well. Intermediate deliverables such as requirements, specifications tend to be uniform reflecting one basic philosophy, since they are the work of one individual (in collaboration with the Backup leader). Often they show hierarchical structure, because they already reflect the division of labor for the Junior Programmers. This role definition reflects the need for few, gifted individuals during the early phases of software development (/MYER76/). When we look at the requirements for the individual team members there are several classes: Highly qualified (both technically and managerially) individuals for the positions of Chief Programmer and Backup Programmer. They have to possess good people skills and be adequate communicators in order to relate well enough to the user (representative) to understand what the user wants. Second they have to be managers who can plan and control, set goals and priorities and assign tasks, evaluate progress and report on progress to the higher management level(s). Third they have to be technical experts in the field of application. They have to be able to decide which aspects need to be investigated for a feasibility study, what the software's functions will be, what the human/machine interface will look like, as well as being able to make the major design decisions. The Junior Programmers on the other hand do not have to be quite so universally gifted. Depending on the level of detail of the design specification they actually may have a very structured, specialized coding task to do with little freedom or room for creativity. Once the work is assigned and specified there is little need for communication and due to the very technical level at which the Junior Programmers work they do not have to have very outstanding communication skills nor the skills involved in requirements analysis or specification writing. Management skills are obviously not needed by them. The programming secretary has to possess skills in Word Processing, Technical Writing, some programming experience as well as the ability to communicate with team members about such issues as change control and document preparation.

From the explanations so far, it is quite obvious that two of the people in this group have to possess rather high level skills in technical and management areas, the Chief programmer and the Backup programmer. The other roles are more specialized. The Junior Programmers are only involved in the implementation phases. The programming secretary fulfills the role of a communicator and standards bearer, again aa specialized function. The adminstrative assistant is, although involved throughout the life cycle, assigned administrative tasks only. If we try to rate these roles in terms of their core job dimensions, then the CP has the highest ratings in all 5 dimensions. This can be very positive for an individual with high needs in all these areas, but can lead to over-activation through stress and to role-overload with the resulting decrease in job satisfaction. The CP also has to be fairly well balanced in his skills and work motivation in all three areas: task, relationships, and influence/direction. If we have an overemphasis on one or a deficiency in another there may be problems. This is clearly pointed out in a critique of the concept (/MCCL81/) which mentions the dangers of (a) expecting too much from the CP, the "Superprogrammer", or (b) having a powerhungry primadonna at the helm (imbalance in the motivational area). Case (a) can be dealt with through task redesign by delegating some of the responsibilities, the most obvious being along major skill boundaries which span the entire software lifecycle: managerial, administrative, and technical. Another possibility is to do this according to phases: requirements and specifications versus design and implementation. Since a considerable amount of communication is involved in either solution it is not a very good idea to separate the functions strictly along one or the other of these dimensions. It leads to too much communication effort between the different "commands" as responsibilities are shifted, unless very strict standards are imposed which depending on the personalities involved may not give them enough of a sense of autonomy to keep them satisfied.

The second danger McClure points out is the lack of checks for the CP's ego. This can basically be dealt with through a similar approach: built-in delegation of authority to make decisions, either rotating decision making power through different phases or splitting it according to task areas: administrative, managerial, technical.

The next question is to whom are these responsibilities delegated? An obvious first solution is to the Backup Programmer. He has to have all the qualifications of the CP, but so far has no guaranteed authority unless voluntarily delegated from the CP. Thus the team concept creates a situation which can create high levels of negative motivation due to a low ranking in the job dimension autonomy. This combined with high motivation levels in the area influence/direction can lead to a potent and lethal situation: why am I working so hard when I do not make a difference anyway? I know as much as the CP, but nobody does what I think should be done. Why bother? In other words the result of this structure can be a marked lack of enthusiasm and resulting lower performance. It should be noted that this need not occur, if the relationship skills (and motivation) of the CP are well developed or if the autonomy needs of the BP are not very high, but not everybody has the talents and personality traits to be a good "vice president" whose role is to know it all, but to stay in the background until something happens.

The Junior Programmers may also lack enthusiasm when their work is "overspecified" for their skills and their needs. A JP may be very happy and content with a precise low level design specification for his/her work at first, but resent the lack of involvement in design decisions, the lack of meaningfulness in his work because he is not given an adequate picture of the whole product, and the lack of communication for his relationship needs at some later time. Again, this may but need not happen, especially if the CP realizes the growth needs of his staff and delegates this part of the work when a JP is ready for it. Unfortunately the CPT does not give guidelines for this. It does provide a good learning environment for JPs when the CP and the BP are actively pursuing making it a good one. Again, there are no guidelines. They have to include sample task assignments for all levels of JPs, since obviously a transition from JP to BP or even CP is only going to be successful, if the JP has been trained in all the areas the CP has to cover. This not only includes the low communication technical tasks, but also the high communication technical tasks such as developing user requirements and specifications. It spans all the technical, administrative and managerial aspects of the entire software development lifecycle. Unless a JP is trained in all of these, he may not be able to develop his/her talents as he needs to (remember, they are known to have high growth needs), nor will he/she be able to prepare adequately for

becoming a BP or a CP. Suggested additional guidelines for training are:
* evaluate growth need and motivations (what do they like to do)
* evaluate major skill areas (what are they capable of or show promise in)
* involve JP to skill level in all of these areas (let JP do some of the work he wants to and can do)
* reevaluate additional needs and motivations (take stock and give feedback to JP)
* add responsibilities in these areas to help the JP grow.

Obviously this needs time and this time has to be built into the schedule. It is fallacious to think that learning takes no time on the job. It does, for the teacher as well as for the student. The idea of creating a good learning environment had been one of the goals of the CPT, but the definition of roles was too static and did not explain adequately the progress of the JP and how it should be dealt with within the team structure. Additional responsibilities should be added in all areas of the tasks involved, on the administrative, managerial and the technical level with the objective to expose the JP over time to all the aspects of work of the CP which now even includes teaching explicitly (another responsibility which can be delegated in degrees). It is also suggested that the JPs are familiarized with the skills of the programming secretary and the project administrator, if they are motivated to learn these skills.

IV. The Surgical Team (ST)

This team structure is very similar to the CPT. Like the CPT it is based on the concept of specialization, but unlike the CPT which enables specialization to the extent of assembly line programming for the JPs with the resulting low level of task identity, task significance, autonomy and possibly feedback, the Surgical Team (ST) specializes such that task identity and task significance still rate high. This is achieved through defining roles for the following areas of specialization: technical, administrative, editing and clerical. In particular the roles of the team members are defined as follows:
1. Surgeon
He is the technical manager much the same as the CP. One significant difference in the ST is that a lot more administrative tasks are delegated to the administrator (who reports to the Surgeon). This narrows the skill variety dimension somewhat, which had been a cause for possible role overload in the CPT.
2. Copilot
The Copilot's responsibilities in the surgical team are the same as the backup programmer's in the CPT. In addition, the Copilot is responsible for interfacing with other teams. This additional function enhances the role of the copilot, takes some of the workload off the Surgeon and gives the copilot some visibility which serves to increase the sense of influence and importance which this role lacks otherwise.
3. Administrator
The administrator is responsible for personel, budget and procurement which includes space, computer time, technical tools, and also interfaces with management. This role requires limited technical expertise, but a good deal of administrative knowledge and qualities, negotiating and planning skills.
4. Editor
The responsibilities of this function are to generate all project documentation. This requires good technical writing skills and communication skills with the other technical members of the team.
5. Secretaries
One or two may be necessary to support the tasks of the administrator and the editor. They are clerical support personnel which need typing and communicative skills.
6. Programming clerk
This role is the same as the programming secretary of the CPT with the same responsibilities except for those which now have been assigned to the Editor.
7. Toolsmith
This is a new role specializing in providing and keeping operational all necessary technical tools such as utilities, libraries, debuggers, etc. Part of this function was unspecified in the previous structure and probably assigned to one or the other of the JPs on an ad hoc basis, part of it was the responsibility of the programming secretary. This role requires the skills of a good systems programmer as well as some communication skills, because even if the editor is responsible for generating the documentation, the toolsmith has to communicate to him/her what needs to go into them.
8. Tester
The function of the tester includes the implementation of the test plan (which is provided by the Surgeon), creating test data, test drivers, debug procedures and the like. It should also include evaluation of tests and feedback to the group how well they are doing in their implementation efforts. Maybe included could be data collection for a software metrics database which can be used for empirically founded cost and schedule estimates as well as quality predictions.

For more detail see /DEMA83/. The function of the tester is one which does not require the constructive ability of implementing a coded solution, rather he/she needs a "destructive" talent to be able to and to enjoy finding errors and knowing how to go about finding them, and as many of them as possible. A patient, perceptive, analytic detail oriented mind is needed for this job. It is not enough to find out that there is something wrong, but also where and what it is. These talents are not the same as those of a good designer or of a reliable coder.

9. Language Lawyer

This individual is the expert on programming languages and knows the most efficient ways to implement the design specifications as well structured code. This definition specifies the function of coding. Again this is a very strictly defined function with a high degree of specialization.

The motives underlying this team structure are to provide a formal structure which enables mangement control for quality, schedule, cost according to goals and priorities. This goal is obviously met very nicely. As the CP in the CPT, the Surgeon in the ST has the power to exercise as much influence and direction as he/she sees fit. The role definitions are even more explicit, and narrow for some, than in the previous team concept. Interaction with the environment is emphasized through the role of the administrator (to management), the co-pilot (to other teams), the editor (roject documentation) and the Surgeon who is still responsible for user interaction. Since the administrator reports to the Surgeon, interfacing with management may have its problems, because the administrator reporting to the Surgeon may not have adequate negotiating power. Product structure will again reflect the cooperation between Surgeon and Copilot in the early phases and result in the hierarchical structure we have seen in the CPT for the very same reasons. We can expect uniform project and product documents, since they are written by the same person. When we look at the needs and requirements of the individual, we can clearly see a reduction in skill variety for all functions involved. This is one of the declared goals of the ST. It has also been pointed out that this can pose motivational difficulties, if one of the specialized tasks is assigned to a person who needs a higher degree of skill variety. Task identity may also be a problem, because the specialized skills of the different functions may not be needed during the entire life cycle and this subtracts from the sense of being involved in a job (i.e. developing a particular piece of software) from beginning to end. Task significance is very high due to the idea to give team members functions which indicate that they are experts in their own right: High task significance is supposed to increase team morale and improve individual recognition. With the specialization the way it is proposed here comes a considerable degree of autonomy in the area of specialization as far as procedures go, but not necessarily with respect to schedule. The degree of communication and cooperation these specialized roles require prevents this. We also have functions defined which may not be perceived as having enough autonomy which actually may be due to overspecialization (not enough skill variety). For example the language lawyer may feel like a coding machine and the toolsmith may not perceive enough connection to the goals of the rest of the team. Feedback is generally good in this structure due to the need to use each other's work. The major potential difficulties in this structure arise due to limited skill variety which may be experienced by the team member as work which is not meaningful enough. The roles of the team members stress complementary skills and thus are enhanced by complementary work motives. The more technical oriented roles require task motivation and depending on the degree of interaction with the environment relationship and/or influence/direction motivation (in the case of the Surgeon). The Surgeon and the Copilot are sharing some of the work more equally now, but the Surgeon may still experience role overload and over activation due to the variety of skills required from him/her. And the degree of autonomy combined with a high motivational level in the influence/direction area may pose problems for the Copilot.

The goal of a good learning environment is remarkably absent from this team definition. And upon investigating the potential for growth and development of the team members it is clear from the definition what that means: the team members are recognized as experts in their respective area and that is it. They are supposedly experts. In other words, the concept of a junior member does not come up. On the other hand Dp professionals have high growth needs (/COUG78/, /FITZ78/), they tend to want to learn new things and tend to get bored, if they have to do the same tasks over and over. One solution is to train them outside for new and now more complex tasks IN THEIR AREA OF SPECIALIZATION or, if the function requires more than one person, to add a trainee who during development can learn under the supervision of the expert. Another issue then is the question how to train a Copilot or a Surgeon. Since all the other roles are so specialized, they do not address the training in the multiplicity of skills required from the Surgeon. The only remedy here is to rotate the prospective Copilot and or Surgeon through the different functions on the team. For some he may only have to serve as an apprentice, for others he may have to acquire skills high enough for the expert position. This ensures that there is a promotional path within a function as well as beyond functions. Adequate promotional paths are considered prime motivators for continued job satisfaction as they provide a pattern for fulfilling growth needs. The concept of apprentices which was added here also serves to increase the sense of autonomy, of being in control for those roles which may be perceived as having very little otherwise (e.g. Copilot, Language Lawyer).

This team concept should work well, when there are two well rounded and highly qualified individuals available who can adequately fulfill the roles of Surgeon and Copilot whereas the rest of the team members have specialized interests and task motivations in the areas defined by the functions above. They have to possess a degree of sophistication commensurate with the complexity of the project. People with a need for high skill variety are not expected to function as well in this type of structure in roles other than Surgeon or Copilot.

V. Revised Chief Programmer Team (RCPT)

This team concept is a further development of the CPT by McClure (/MCCL81/) which tries to remedy the following perceived shortcomings of the CPT:
* role overload of CP
* inadequate level of autonomy of BP
* environment not open and sharing enough
* project too dependent on individual team members (i.e. CP)
* not enough visibility for team to the outside
* inadequate degree of formalization of individual's responsibilities
* CP has too much power
She tries to overcome these shortcomings mainly by redesigning the tasks and responsibilities of the CP and the BP, reducing the areas of responsibilities of the CP by creating two new positions, that of the user liaison and that of the administrator who takes over all the administrative tasks of the CP with the administrative power (the CP now reports to the adminstrator). The user liaison also takes over some of the responsibilities of the CP including all direct dealings with the user. The coleader's role is enhanced through additional areas of prime responsibilities notably representation of the team to the outside and coordination of project turnover with the maintenance group. He also now has the responsibility of developing the test plan. The CP only reviews it.

If we compare this structure to the previous two, then we can clearly see that some of the problems of the CPT and the ST have been overcome. Work and recognition is more evenly spread. At the same time there is enough structure in this team to enable adequate management control. As a matter of fact the tasks are more precisely defined in this team concept for the roles of Surgeon/CP and Copilot/BP than in the other concepts. Interaction with the environment is a lot more emphasized than before by creating two new positions, the administrator (interface to management) and the user liaison (interface to the user) and explicitly mentioning the need for communication to other teams. Tending to those needs is the express responsibility of the Coleader. Maintenance preparation is another issue which is not explicitly addressed in the other team structures, but here it is the Coleader's job. The team, since it is not as strictly hierarchically structured should adapt to a wide variety of posible product structures. Through the primary involvement of mainly three people, the leader, coleader and the user liaison, one can still expect a fairly unified product developed in a consistent design philosophy. The function of user liaison ensures that the product will actually meet the user's needs. Looking at the requirements and needs of the individual, we see that the roles of the four nucleus positions no longer require the same skill variety as in the CPT or the ST. As a matter of fact the four positions of administrator, project leader, coleader and user liaison are created by following the concept of division of labor and specialization. All technical tasks are the resonsibility of the project leader and coleader, the administrator only handles managerial and administrative functions and the user liaison specializes in user/developer communication. For all these positions this means a reduction of skill variety compared to the CPT. The ST already had made a step into this direction, but mostly for the roles other than Surgeon and Copilot. Thus the ST and the RCPT differ markedly in the direction of specialization: The RCPT reduces skill variety for the positions most critical to the first phases of the software development lifecycle, whereas the ST specializes the programmer positions, but leaves the others with a much higher degree of skill variety. The RCPT mentions similar functions as the ST does, but it does not mandate whether they should be done by a "specialist" or by several people together with a resulting higher skill variety for everybody. This obviously increases this concept's potential for skill variety as well as task identity (which is true for the four nucleus positions anyway). Task significance is higher in this concept than in the previous ones, provided that the programmers are not assigned to the specialist roles of the ST. The RCPT gives the project leader considerably less autonomy in the administrative aspects of the work, but still leaves freedom for schedules and procedures. Feedback between the members of the nucleus is expected to be fairly high, since they work on a preliminary product together (requirements, design), before the final product is ready for release. The amount of feedback for the programmers is uncertain and depends on the actual work assignments and control mechanisms chosen which are left unspecified in this concept. As a result this concept has a good potential to structure work such that it is experienced as meaningful, that the team members feel they are responsible for work outcomes and possibly that they know about the results of their effort. To ensure this, it is suggested that someone from the development team is involved in product maintenance once the software has been delivered. Why that is supposed to give feedback to ALL members of the development team is unclear, however.

Due to the degree of specialization of the members of the nucleus we need people with different areas of skills and motivation, managerial skills and influence/direction motivation as well as relationship motivation for the administrator. Primarily task motivation and some relationship motivation is needed for the leader and the coleader, relationship and task motivation for the user liaison, and task motivation for the programmers. The skills required are managerial and administrative for the administrator, technical and communication skills for the project leader, and the coleader, and a great deal of communication and negotiation skills and some technical knowledge for the user liaison, technical skills for the programmers. If we have people with these qualifications we can use the RCPT to advantage.

Once again, this team concept does not mention training and professional growth as part of the concept. It is possible however, to use the suggestions made for the previous two team structures. Of all the team structures reviewed this seems to be the most balanced one (consequently suited for the most balanced set of people).

VI. Egoless Programming Team (EPT)

This is the last team concept to be reviewed. It works on the basis of free cooperation with no specific roles or reporting structure within the team. Everybody is responsible for everything. The team works towards a common team goal in a totally democratic work environment. If there is a leader or if there are assigned rules, they have been agreed upon by the majority of the team members and are only assigned "until further notice" when a subsequent vote changes the assignments. Thus team leadership may rotate. So can function. The idea behind this concept is to give full autonomy to the members of the team. In other words there is no formalization of team structure which enables management control. It is thought to reinforce team spirit similar to the Volvo experiments (/GYLL77/, /FOY76/). This nonhostile environment is hoped to be an excellent learning environment because everybody is involved in everything. One other reason for advocating this team structure is, that it is dangerous and unproductive to allow programmers to "sit on their code", because it causes them to regard it as extensions of their egos, resulting in tunnel vision and more undiscovered bugs. Also, since everybody is involved equally, the result should be a better integrated system. Code exchange is mandated as an important part of the development with the hope of having a more visible, better readable and more reliable system. As mentioned before, this concept does not provide adequate management control. This concept, also called "autonomous work groups", may have worked in auto production where tasks are well defined and repetitive, but the developing of software does not have these properties, thus increasing the complexity of the work significantly. Software developments also tend to take much longer than putting a car together.

The democratic approach often proves to be much too loose for management control, not only because the performance of the individual cannot be evaluated easily, but also because, since nobody has the decision making power to settle disputes (over design decisions for example), the whole team can turn into a debating club where no work gets done. Decision making may be postponed indefinitely. During a crisis when leadership is needed, it is often hard to find somebody who is willing to take it over.

Visibility of the development team to the outside is another problem. There is no one person to whom the user should talk to, there is no one person who is in charge of communicating with other teams or with management. This can be very confusing and frustrating for the people involved. Product structure again tends to parallel the (informal) team structure which now depends on how team members relate to each other. The earlier deliverables such as requirements and design specifications probably will not be quite as uniform, since a lot more people will be involved and a lot more different opinions need to be reconciled and integrated.

The individuals in a team structure like this need to be able to deal with high skill variety or be able to negotiate a task commensurate with their inclinations and talents. Otherwise they may end up confused, overstressed and as a result experience motivational problems. Since everybody works very closely together and is involved in every aspect of the software development process, task identity and task significance are high. Autonomy also is rather high to the point that team members may become disoriented, confused, or unmotivated, because "nobody is telling them what to do". Feedback is built into the system through working so closely with others. As a result this team concept lends itself to have its members experience the meaningfulness of their work and knowledge of the results. It tends to have problems in the area of perceived responsibility for work outcomes of the individual. People with strong task motivation may be very happy in this environment, but they may also experience a lot of frustration when they feel that they know best, but others don't agree and they lack the relationship motivation and the negotiating skills to deal with thc situation. All members have to be relationship motivated to a degree. If there are too many people who have significant influence/direction motivation, they may all strive for being the group appointed leader and severe conflicts may result. This concept seems to work best when thc group is small, because there are less people to communicate and negotiate with, when the members acknowledge each other as cqua1s, and when they have enough expertise and are goal oriented enough to be able to set and achieve their own objectives.

The claim that this team concept provides an excellent learning environment is only partially justified. A "new kid on the block" may in the beginning pose more of a problem than be an asset. Often groups "isolate" unproductive members by giving them tasks which do not have a lot of impact on the group's success, thus pushing the member to the periphery. This avoids the extra teaching and communication effort and reduces the risk for the rest of the team. It does not motivate the junior person a whole lot though (low task significance). A better idea is to have a mentor for the trainee or convince the group that besides developing software they ALSO have the goal of educating the junior member and making this task just as much a team objective as software development itself. To achieve this there must be a clear incentive (reward) for the team, e.g. a mentor reward (competition) and/or a monetary incentive for the group, based on the relative learning achievement of the trainee. Teaching may be done by rotating mentorship, so that everybody gets a chance to be the teacher.

In conclusion, the egoless programming team has some advantages over the other teams in terms of autonomy for the members, but due to its poor management controls and high need for interaction and communication, it should only be used for very small teams where the people involved relate well to each other. This obviously limits the size of the project which can be done considerably. It is a disastrous idea to use this concept with people who need close supervision or are not able to set realistic goals for themselves.

VII. Conclusion

This paper attempted to show how the most commonly advocated team structures can be used, what their characteristics are, where their advantages and disadvantages lie and how to select a proper team structure based on work and people characteristics. Some of the answers this paper tried to provide have been adapted from methods for task design (a good textbook is /GRIF82/). Because of space limitations, thc problem of task and employee evaluation was dealt with in a general way. However, there are job analysis questionnaires and core job dimension analysis tools available (see chapter 5 in /GRIF82/) as well as methods for measuring motivation levels in the areas of task, relationship and influence/direction. Specific evaluation instruments for members of a software development team are basically nonexistent. However, the more general instruments are quite suitable for this purpose if one is willing to accept that it will require somewhat more effort to use it than if one ahd a more specialized evaluation instrument. At this time no experimental results are known to the author which evaluate projects, their strenghts and weaknesses as a function of team structure and team member characteristics and provide a statistical basis for choosing team structures. It is hoped that through joint efforts with industry a data base like this can be developed which can further broaden our understanding of software development as a team effort.

References

/BABB32/ Babbage, C., "On the Economy of Machinery and Manufactures", Charles Knight, London, 1832.

/BAKE72/ Baker, Mills, H., "Chief Programmer Teams", IBM Systems Journal 11, (1), 1972, p. 56-73.

/BROO75/ Brooks, F. "The Mythical Man Month", Addison-Wesley, 1975.

/COUG78/ Couger, J.D., Zawacki, R.A., "What Motivates DP Professionals?", Datamation, Sept. 1978, p. 116-123.

/DEMA83/ DeMarco, T., "Controlling Software Projects - Management, Measurement and Estimation", Yourdon Press, 1983.

/FITZ78/ Fitz-enz, J., "Who is the DP Professional?", Datamation, Sept. 1978, p. 125-128.

/FOY76/ Foy, N., Gadon, H., "Worker Participation: Contrasts in Three Countries", Harvard Business Review, May-June 1976, p. 71-83.

/GRIF82/ Griffin, R.W., "Task Design - An Integrative Approach", Scott Foresman, 1982.

/GYLL77/ Gyllenhammar, P.G., "People at Work", Addison-Wesley, 1977. Addison-Wesley, 1975.

/HACK76/ Hackman, J.R., Oldham, G.R., "Motivation through the Design of Work", Organizational Behavior and Human Performance 16, 1976, p. 250-279.

/HACK80/ Hackman, J.R., Oldham, G.R., "Work Redesign", Addison-Wesley, 1980.

/MCCL/ McClure, C., "Managing Software Development and Maintenance", Van Nostrand Reinhold, 1981.

/MILL83/ Mills, H., "Software Productivity", Little, Brown & CO., 1983.

/MYER76/ Myers, G. "Software Reliability: Principles and Practices", Wiley, 1976.

/SMIT76/ Smith, A., "An Inquiry into the Nature and Causes of the Wealth of Nations", Modern Library, New York, 1936. Originally published in 1776.

/STEV77/ Stevens, H.P., Krochmal, J.J., "Engineering Motivators and Demotivators", IEEE 1977 Natl. Aerospace and Electronics Conference, p.162-168.

/TAYL11/ Taylor, F.W., "The Principles of Scientific Management", Harper and Row, 1911.

/THAY82/ Thayer, R.H., et al., "Validating Solutions to Major Problems in Software Engineering Project Management", Computer, August 1982, p. 65-77.

/WEIN71/ Weinberg, G.M., "The Psychology of Computer Programming", Van Nostrand Reinhold, 1971.

/ZOLN82/ Zolnowski, J.C., Ting, P.D., "An Insider's Survey on Software Development", 6th Intl. Conf. on Software Engineering, Tokyo, Japan, Sept. 14-16, 1982.

SESSION OVERVIEW
MANAGING END USER COMPUTING

Thomas N. Pyke, Jr.

National Bureau of Standards
Washington, D. C. 20234

We are now in an environment in which we observe the proliferation of small computers due to their rapidly decreasing cost along with continually increasing capabilities. Fueled by vendor and media promises, direct access to computing resources by end users, including use of micros, is becoming widespread. Many end users are requesting or even demanding increased and improved access and relevant support from management. Many in management are encouraged by these developments and would like to further them, but are increasingly aware that there is a growing need to impose appropriate organizational constraints.

The objective of this session is to explore and summarize the issues and motivations in implementing an integrated approach to supporting end user direct access to computing resources. This is done by providing views from three perspectives: historical, user, and management.

The first talk will present a historical view of the evolving information resource center (IRC) concept, provide a set of working definitions, and give focus to the central themes involved. It will explore the objectives that may be achieved by means of an IRC program and identify those who are likely to be the driving forces behind and advocates of such efforts. The talk distinguishes between various types of support, including handholding of end users of mainframe services and establishing microcomputer support centers. It reports on activities underway to integrate such support and implement the various components of such an IRC program. It will also distinguish between the "helping" vs "controlling" nature associated with an IRC.

The second presentation, "Information Center - The User's Answer to the Computer Room," will look at support structures from a user's point of view. As the end users sense the potential of the new technology in improving their individual performance and productivity, what types of help in realizing this potential are they seeking? What kinds of computing resources do they need? What help do they need in identifying and evaluating the information alternatives available to them? These and other questions will be explored.

The final presentation "Support Structures: A Management Tool," will advance a management perspective on support structures. The view of these structures as a vehicle by which management can foster individual productivity on an incentive basis and at the same time influence the direction of their use will be examined. The role of support structures to ensure adherence to organizational constraints (which address such considerations as data integrity, auditability, and security) will also be explored.

INFORMATION CENTERS: THE USER'S ANSWER TO THE COMPUTER ROOM

Esther P. Georgatos

Veterans Administration
Office of Data Management and Telecommunications
Washington, D.C. 20420

ABSTRACT: New "Information Centers," which utilize and promote personal computer use, have gained popularity in many large businesses and are now finding their way into the Federal Government. Their most interesting feature is that the users operate the equipment themselves. While the Centers aren't capable of doing the large jobs currently handled by the typical DP department, they are introducing the user to personal computing and to more advanced data processing theories. This paper briefly describes the Information Center concept and discusses the establishment of a Center at the Veterans Administration in Washington, D.C.

KEY WORDS: Information Center, DP department, personal computers, users, data bases, user needs, Implementation Plan, work environment, Information Technology Center (ITC), staffing, publicity, stand-alone, data manipulation, networking, testing, modifications, office automation

1. Introduction

About a year ago, COMPUTERWORLD ran a cartoon which illustrated how the role of the user has evolved through the sixties, seventies, and eighties. It depicted the DP department in the sixties as monarch over the entire computer environment. The user was little more than a faithful subject. But in the eighties, or so the cartoon predicts, the user will be the one holding the keys to the computer room--and the DP department will be pretty much left out in the cold.

Whether this prediction ever comes to pass is arguable at best. But without question, the user's growing involvement in his or her computer needs is a trend that is here to stay. In fact, the smart DP departments have already begun abdicating their absolute authority in favor of more democratic arrangements.

One of the most successful of these arrangements involves the operation of a self-contained unit called an "Information Center." While its introduction can't be heralded as the final solution, it does seem to be an idea whose time has come.

2. The Concept

An Information Center is an independent organization that uses the personal computer as its primary tool. While it normally operates within the DP department, many users are establishing Centers within their own organizations--especially when the DP department has been slow to act.

The Center's main purpose is to provide the user with a means to accomplish smaller jobs not easily handled by the organization's huge computer systems. Its strength lies in its ability to produce a product quickly and easily, and to deliver results that almost guarantee user satisfaction. This third seemingly impossible task can be accomplished because the Information Center is unique in one important way: The actual work is done by the users themselves.

A second important purpose of the Information Center is to provide a systematic approach to the growing arena of personal computing. The very nature of the Information Center's imposed structure allows users to purchase their own equipment, but at the same time keeps them within a prescribed structure. This allows for networking capabilities and consistency when outside services are used or new applications are added. In essence, it puts the DP department in a guidance role rather than an operating one.

Finally, so as not to forget the user's current needs, the concept still leaves the DP department with its large global systems, entities that meet needs not possible with personal computers.

3. The Approach

While the concept of the Information Center is relatively new to the government, it is not new to business or academia. Corporations have used the concept for several years to bolster productivity and reduce applications backlogs. Colleges and universities have employed the concept even longer, having begun making their data processing resources available to students back in the sixties.[1]

The success these organizations have achieved has not only verified the validity of the concept, it has also helped define the approaches Centers are likely to follow.

Generally speaking, Information Centers fall into two basic categories: The first kind provides users with access to the organization's data bases; the second uses several types of personal computers as stand-alone units. Both, however, have the same responsibilities:

- Establishing the user-friendly environment required of the concept
- Selecting whatever software tools are appropriate
- Ensuring that the proper data is accessible and secure
- Acting as consultants to the user community on an as-needed basis[2]

In addition to these responsibilities, most Information Centers have taken on the task of training the users. This is not only something of a necessity, it is a good way of establishing good rapport between the DP department and the user.

Most Centers also operate a library of trade publications, software packages, and vendor literature. Others provide space and plug-ins for users who want hands-on access to vendor systems they are evaluating for procurement. Still others provide areas where vendors can display and demonstrate their products.

4. The Implementation

As a means of discussing how a Federal agency might go about implementing an Information Center, it was suggested that we describe our own experiences in establishing a Center at the Veterans Administration in Washington, D.C.

Since there are only a few examples of an agency setting up an Information Center without the help of outside consultants, we hope this documentation will prove useful to our Federal associates who are considering a Center of their own.

4.1 Inception

The decision to establish a VA Information Center was made initially in February 1982. It was decided that it should be located within the DP department, in this case, the Office of Data Management and Telecommunications. The physical location of the Center was to be the VA's Central Office in downtown Washington.

4.2 User Needs

Regardless of the agency function, every Information Center must lay the user's needs as its functional cornerstone. These needs determine how the Center should be constructed, in terms of both equipment and size.

The needs of our users called for a combination of the two types of Centers discussed in Section 3. That is, access to agency data bases was required, in addition to the services only stand-alone systems could provide. Equipment was thus procured that provided for data base access, computer graphics and statistical modelling, the display of plotted information, and interactive processing of data.

As for size, the Center not only had to house the equipment, it also had to be large enough to meet other user needs, e.g., vendor demonstrations, the training of user personnel, and the operation of a personal computing library.

4.3 Implementation Plan

After we had defined the user's requirements, we began translating them into an Implementation Plan, with the intention of meeting the user's highest priority needs first. However, we soon ran into several stumbling blocks which made a specific, highly detailed plan difficult to formulate.

For instance, all electrical work and carpentry had to be handled by GSA, the lessor of the VA building. That meant our renovations were completed according to their schedule, not

[1] "The New Info Centers," DATAMATION (August 1983), p. 30

[2] "System Development Mythology," DATAMATION (August 1983), p. 276

ours. Consequently, the plan couldn't be structured towards the Center's opening, since that date was not known. Even the staff could not be brought together, since the area it was to share was still occupied by other employees.

But in spite of these difficulties, the Implementation Plan still proved useful. It was the rallying point of all efforts and the means by which schedules and deadlines were kept track of. Even when delays occurred, we knew where we were going--if not necessarily when we would get there.

4.4 Work Environment

Naturally, the physical shape of the Center had to adhere to the room in which it was to be placed. As can be seen by the floor plan (see Figure 1.), our room was large enough, but it was rather oddly-shaped. In addition, it was poorly lit and in a state of disrepair. Our first priority, then, was to determine how best to transform this space into a pleasant, practical working environment.

We decided that the Information Technology Center (as it was formally named) could be divided into seven distinct functional areas, grouped by use. These included the following:

- Office automation
- Computer graphics
- Library
- Reception area
- Time sharing
- Personal computing
- Plotting

The room's odd shape lent itself very well to this concept, as we were able to place the equipment in separate areas and still have room left for the library and reception area. In addition, the middle could be left for demonstrations, seminars, and training. We also had space available for future needs.

After the floor plan was conceived, attention was turned to more practical items. To begin with, the existing chilled air cooling of the room was not adequate, considering the space we had (1000 sq. ft.) and the amount of equipment we would be operating. Consequently, new air conditioning ducts were added. A new ceiling was also hung, to accommodate additional light fixtures. A separate power supply was set up to eliminate voltage deviations in the laser printer.

After these matters were taken care of, attention was turned to the overall tone of the work environment. We wanted to make the room light, airy, and as esthetically pleasing as possible. Furniture was chosen to re-enforce this theme, although comfort and practicality were the final determining factors. In addition, the room was ringed with an electrical cable raceway, to conceal wires and to keep them from under foot.

The resulting effect was one of freedom and open space, perfect for creating a helpful atmosphere and for enhancing user confidence and access.

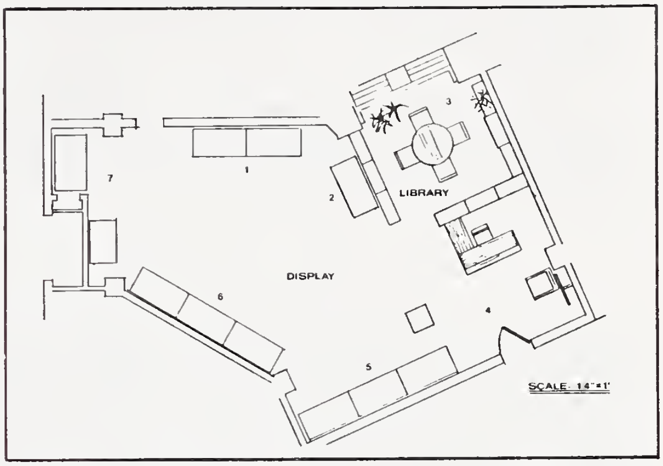


Figure 1. Floor Plan of the VA Information Technology Center (ITC)

4.5 Staffing

The most critical part of our implementation was the hiring of the staff. Again, our user's needs were the standards by which personnel were chosen. Since the ITC is geared more toward technical consulting than training, a staff was selected to reflect expertise in technical areas, including programming, systems and statistical analysis, modelling, and telecommunications.

Placing the emphasis on the technical side will also allow us to more easily keep abreast of the new technologies constantly being evaluated, modified, and developed by the Center. It has also proved useful in setting up and maintaining new equipment.

4.6 Publicity

It has been said that word-of-mouth advertising is the best way to publicize a product or service, since it offers the advantage of instant credibility. But in order for it to be effective, you must have a lot of satisfied customers. Fortunately, in the few months we operated before renovations began, the ITC was in constant use, and most importantly, users were excited about the concept. From the beginning, we emphasized that the Center existed for their use, and we still try to get our users as involved with its success as possible.

However, we also decided it was important to use a more expansive publicity program, one that would blanket all of the user organizations and locations. To this end, a brochure was written that outlined the ITC's purpose, services, location, and use. We also instituted a bi-monthly newsletter, providing the user with information on new techniques, new systems, new services, and possible problem areas. Finally, we wrote articles in various VA publications, describing the services offered and the concept in general.

All in all, we have been able to cultivate some positive opinions about the ITC, which in turn has kept us encouraged and especially alert for ways to improve our services.

4.7 Current Status

In less than nine months, the ITC has been staffed and trained, renovations have been completed, and host connections are in place. Most of the equipment is already in use, and the rest is on order. While we're not yet totally operational, users have already begun signing up for demonstrations and training. The ribbon-cutting is scheduled for November of this year.

4.8 Future Plans

While most of our efforts have centered on immediate needs, we have not lost sight of our responsibility to accommodate the growth and sophistication of the user. A good foundation has been laid, and we are looking forward to some exciting possibilities.

To begin with, our users will soon be using existing stand-alone units as part of an interconnected network. This will enable them to supply their own data and then to create a variety of ways to display it. The next stage will find the user accessing data bases through telecommunications and preparing graphs and other data on-line.

As for the potential of microcomputers, users are already employing software packages to manipulate data. Next will come provisions for the user to access a data base, make an extraction, manipulate the data using one of the packages on the micro, and send the data points either to the graphics system, the plotter, or the micro's graphics capabilities.

As our users begin buying personal computers for their own organizations, our staff can assist them in configuring their systems, selecting software, and helping them with problems they might encounter. As the number of these users grows, the ITC will sponsor user's groups and a bulletin board service, which will interconnect users in Washington with those in field stations.

We also intend to evaluate, test, and modify equipment and software coming on the market. One of our first efforts will involve creating a local area network by linking our microcomputers. This is what we'll use to run our office automation system. It is a brand new concept and will take a lot of testing and modification to make it work.

Research will also begin on the design and implementation of a multi-user microcomputer. This project will use products from a variety of different vendors and will require the staff to find a way to link the products into a workable system. However, it is an important concept, since the result will give us a very versatile system, and one that can use existing equipment.

5. Conclusions

As we previously mentioned, very little documentation exists for setting up an Information Center in the Federal government. While this paper is not sufficiently detailed to provide that documentation, there are a few lessons we learned that might help interested organizations sidestep some problems. These are as follows:

- o Be User Oriented - Establish the services you are to offer in line with the functions of your agency and your user's needs. Once this is determined, be single-minded in getting the staff, equipment, software, and facilities to support your services. Also, realize that everything should be planned or done in response to an existing or anticipated user requirement. Keep in mind that the Center exists to meet the needs of the user.

- o Define and Restrict Your Services - Unless there is an unlimited staff, restrict the use of the Center to the services you are providing. Stay away from complicated projects that are part of the normal data processing function.

- o Train the User Well - Be prepared to meet the needs of the user from an educational standpoint. Users will require a lot of technical support, especially in the beginning. No package has instructions so well written that an untutored user can learn without aid. It is also a good idea to have initial user instructions taped near each piece of equipment.

- o Prepare an Implementation Plan - Time spent preparing an Implementation Plan, however difficult it might be to follow, is well worth the effort. Resist the temptation to stray from it. In order to be successful, all the details must be accounted for. In addition, don't forget about the descriptions of services, policies, procedures, and standards you'll need before going operational.

For those of you who are already convinced of the advantages offered by an Information Center, we hope we have expanded your understanding of the concept and perhaps opened your eyes to some of the exciting possibilities it can offer.

On the other hand, for those managers in today's DP departments who still think of personal computers as over-sized video games, you'd better reconsider. Yours may be the monarchy left out in the cold.

We are especially grateful to Mr. Jack Sharkey, Director of the Office of Data Management and Telecommunications. Without his support and belief in the Information Center concept, the ITC could not have proven so successful.

SESSION OVERVIEW
MICROS AND THE NEW CPE ENVIRONMENT

Dennis M. Gilbert

National Bureau of Standards
Washington, DC 20234

Over the past decade we have seen the field of computer performance evaluation (CPE) parallel the maturity and changes taking place in the computer industry. We have seen the CPE activity expand beyond the more parochial interest in and concentration on "tools" and hardware. Attention has broadened to include concerns about selection and management of information resources, human interfaces, individual and unit productivity. The weight of attention is shifting from component efficiency to organizational effectiveness.

The pace of change in the computer industry has begun to accelerate within the past two years. Rapid advances in the technology, significant improvements in cost/performance, dramatic proliferation, and growing market place and user maturity has made microcomputer-based systems major resources that must be considered as organizations seek to satisfy their information requirements.

In this session we will identify some of the changes the CPE community is and will be facing, look at the likely implications of these changes for us, and explore how we might best position ourselves so as to creatively use change rather than be left behind by it.

The first speaker will offer an organizational view of the infusion of the new technology and present a framework for a new approach to thinking about such issues. The second speaker will address the growing sophistication of users in stating their requirements and explore such developments as integrated environments and interfacing workstations with mainframes. The last speaker will look at some of the new technologies and market developments (eg, videotex, video disc, smart phones, etc) and explore how the convergence of these and others with those of microcomputers may impact the way we do business internally and deliver products and services.

AN ORGANIZATION MODEL AND CASE STUDY FOR MICROCOMPUTER CPE

Malcolm Campbell

State of Missouri
Division of EDP Coordination
P.O Box 809, Capitol Building
Jefferson City, Missouri 65102

In this paper, computer performance evaluation is viewed from the perspective of assimilating the microcomputer into the organization.

It presents a way of thinking about problems and answers. It does this by presenting a model and some sample components of the model.

The approach is an attempt to fit some microcomputer issues into the framework of organization development styles.

Key words: End user; microcomputer; microcomputer laboratory; model; objective oriented management; organization development; organizational tensions; productivity; reference system; team work; technology.

OUTLINE

1. The Challenge of Micro Infusion Into the Organization

This paper assumes that the micro infusion is not an open and shut case. There seems to be agreement that micros are being placed in organizations at a dazzling pace. However, there seem to be valid considerations both supporting and opposing aspects of what is happening. And there are many differing conditions impacting what should be done next. These involve cost, system reliability and the self-determination of the end-user.

This paper explores a way of thinking about infusion questions. A suggested rationale is expressed in terms of a model, application of the model, and case study descriptions for applying parts of the model.

Before getting to the model, what are the kinds of issues we need to look at? Where the micro is a blessing and where it is a curse relates to organization objectives in both tactical and strategic areas. Tactically, acquisition and application to specific tasks raise questions of product choices, system planning, and expenditure of resources. Strategically, the question is how the micro is conceived as a tool and how it fits into organization dynamics.

It is often expressed that white collar productivity has yet to be impacted substantially by technology. Equipment expenditures per white collar employee are only a fraction of that per blue collar employee. However, is expenditure alone enough? Must not application of technology deal effectively with the success of the organization and with the factors of integration and of tension?

How do we pursue integration while encouraging tools which foster tension? Assimilation of microcomputers into the organization relieves some tensions and stimulates other tensions. Let's try, in a moment, to think of some examples of tension. Management of such tensions should direct expenditure of resources toward the goals of the organization. Micros must be assimilated with goals like the following in mind:

1. Maximization of productivity
2. Avoidance of waste
3. Maximization of profitability
4. Maintenance of data integrity

You probably think of some other goals.

To achieve the goals of the organization, integration of the elements of technology, human factors, and management action is necessary. What kinds of models can be used for this? What kind of thinking is needed?

This paper suggests the following:

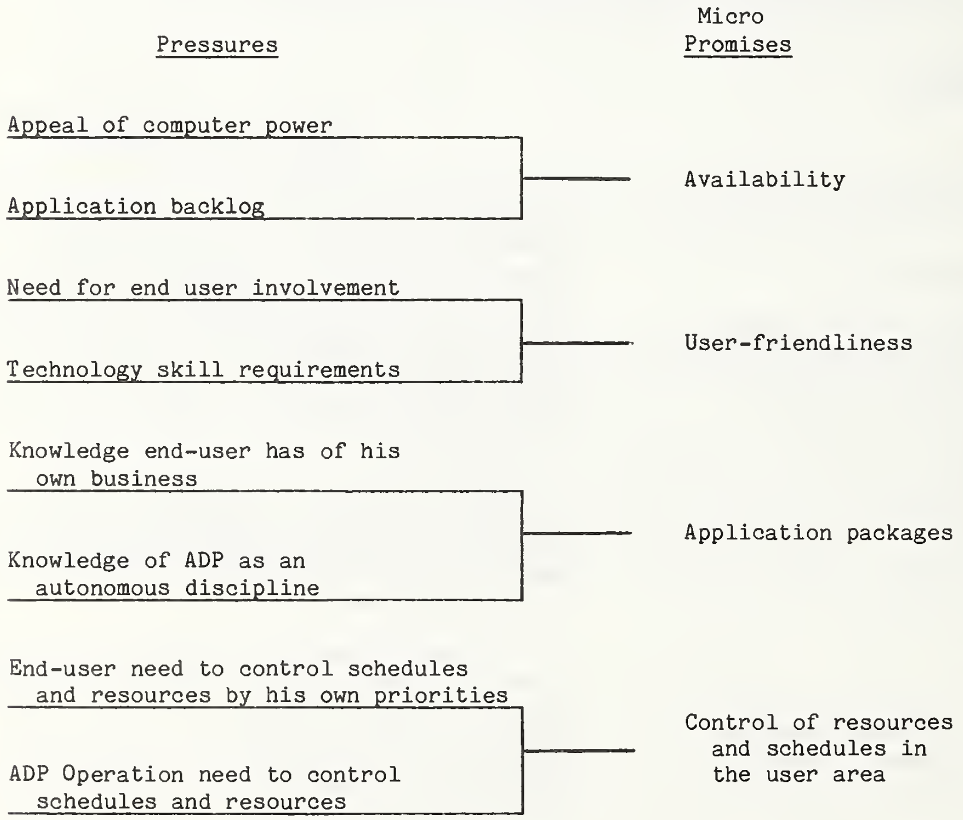
1. An organizational model
2. Examples of integration; illustrations of the dynamics involved.
3. Two case studies illustrating constructs of the model. We'll discuss "constructs" later; for now, they are "organizational tools", or "frames of reference."

2. Tensions in the Organization

Can the model we develop accomodate tensions, integration and goal-seeking? My examples and constructs are an attempt to apply the model to micro infusion.

Before describing the model, what are some tensions in the organization? First, tensions which have led to microcomputer infusion and second, what are tensions which have resulted? Are there models to suggest how to view these tensions in a constructive way?

2.1 Tensions Encouraging Micro Infusion



Pressures

Micro Promises

Appeal of computer power

Application backlog

Availability

Need for end user involvement

Technology skill requirements

User-friendliness

Knowledge end-user has of his own business

Knowledge of ADP as an autonomous discipline

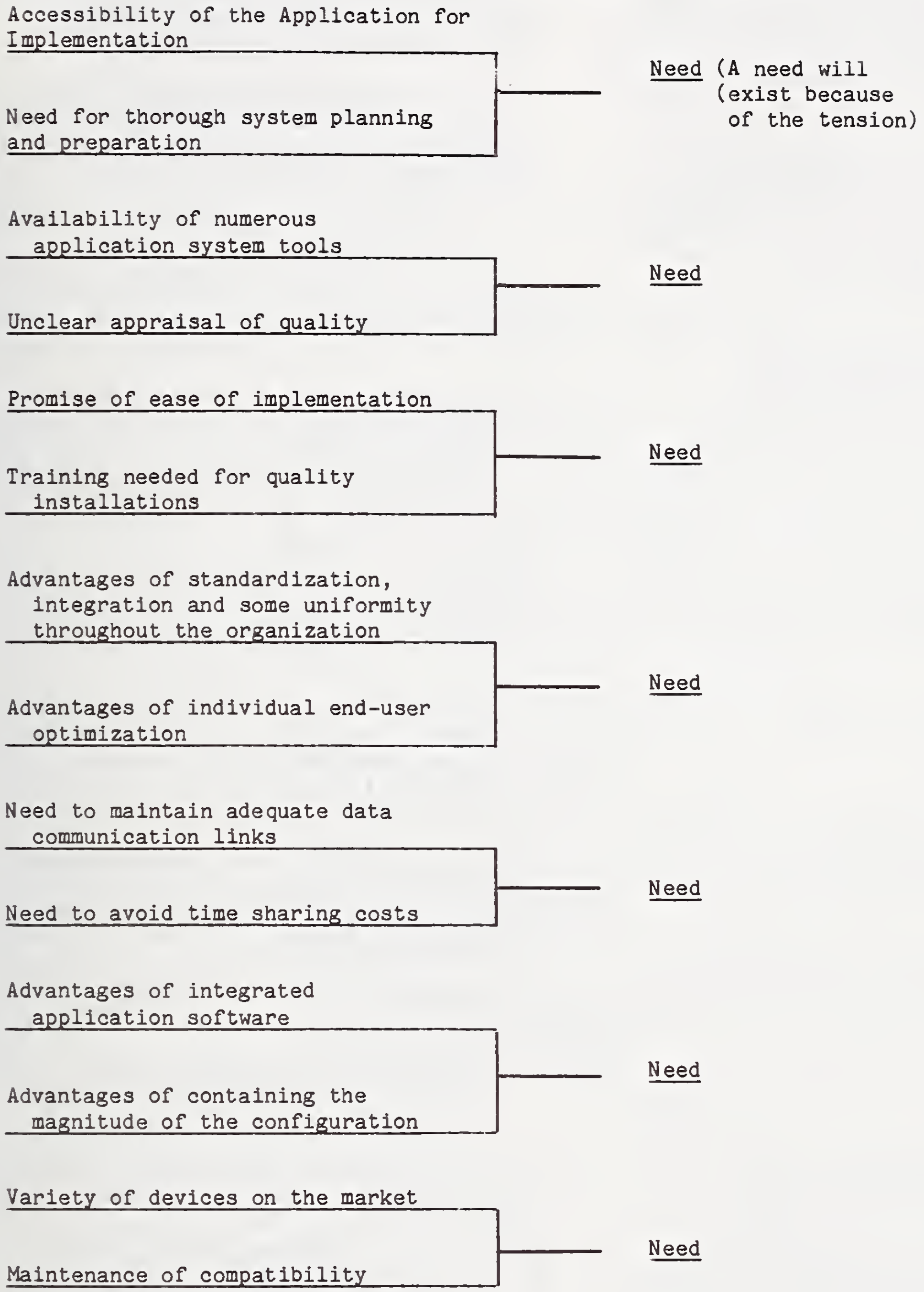
Application packages

End-user need to control schedules and resources by his own priorities

ADP Operation need to control schedules and resources

Control of resources and schedules in the user area

2.2 Tensions Resulting From Micro Infusion



Accessibility of the Application for Implementation

Need for thorough system planning and preparation

Need (A need will (exist because of the tension)

Availability of numerous application system tools

Unclear appraisal of quality

Need

Promise of ease of implementation

Training needed for quality installations

Need

Advantages of standardization, integration and some uniformity throughout the organization

Advantages of individual end-user optimization

Need

Need to maintain adequate data communication links

Need to avoid time sharing costs

Need

Advantages of integrated application software

Advantages of containing the magnitude of the configuration

Need

Variety of devices on the market

Maintenance of compatibility

Need

These needs require work. For example, consider the evaluation of integrated, multi-task, executive productivity packages.

One study asserts that to be able to distinguish the important differences between "MBA" (from Context Management Systems) and "1-2-3" (from Lotus Development Corporation) requires weeks of concentrated study even by a computer-literate manager. That kind of effort is costly, but isn't it also good for the organization?

Tensions in the work-life can give energy to the enterprise. Contradictions, cross-purposes, disparities, paradoxes, and exceptional observations provide sparks that may lead to discovery. Such tensions may suggest that we are looking at something the wrong way. It is by this route that innovation becomes a tool for problem solving. Sometimes undoing faulty integrations may be half the game. A prerequisite for originality is the art of forgetting, at the right time, what we already "know."

So tensions can be used constructively.

Now how does the model relate to this? Modeling is a way of looking at things. It is a way of turning noise into information. It provides the rationale for action in a mixed and changing environment.

Is not how you look at things important? Poincare was Einstein's senior by 25 years. Poincare had the essential facts for the general theory of relativity before Einstein did. Einstein made the discovery because he had the way of looking at things.

So we need a way of looking at things which accounts for tensions, goals, and frames of reference. A model, I suggest, is the "Dialectical Organization Model."

The Dialectical Organization Model provides a way of looking at the tensions of micro infusion and fitting them into organization development.

In practice, application of the model follows a cycle like the following.

3. The Dialectical Organization Model

3.1.1	Recognize the "Squeaky Wheel"	A person with power to have results, becomes aware of conditions needing attention.	TENSIONS
3.1.2	Review Goals	In order to diagnose the degree, nature, location and scope of need.	GOALS
3.1.3	Diagnose tensions or inactivity	Determine whether the squeak was from tension or from inactivity (equilibrium).	DIAGNOSIS
3.1.4	Zoom to bring territory into focus	Select object area. If inactivity, zoom to expose. If tensions, zoom to integrate. Aim is to prepare opportunity to achieve growth toward goals.	ZOOMING INTEGRATION EXPOSURE
3.1.5	Apply constructs for interpretation and action	Reference systems, beliefs, laws, principles, perspectives, criteria, environmental frameworks, pictures, models.	CONSTRUCTS
3.1.6	Experience and monitor results	Progress toward goals organizational learning, growth, integration, new tensions.	GROWTH
3.1.7	Go to 3.1.2		

Above is the operational sequence of the model. I'm sure you have experienced the processes.

Tensions provide the driving forces.

Goals provide the sense of direction.

Zooming focuses, as with a camera.

Constructs provide the rationale.

Integration is sometimes the productivity result, and sometimes new tensions are.

Growth is where value is achieved.

The dialectical sense of the process is made meaningful by the goals and constructs.

You may visualize a number of ways these processes fit the micro world.

4. Examples of Integrations

One of the noticeable outcomes that can result from applying constructs to the tension resolution cycle is integration. You've seen micro situations where integration is appropriate. Here are examples of organizational integration emerging from out of the micro context.

The four examples I have picked are:

1. Fact-based management
2. Manager computer literacy
3. The pro-user
4. Re-defined information

4.1 Management Methods

Contribution of Micros to Fact-based Objective Oriented Management

Perhaps you have observed ways in which micros impact management approaches. With spreadsheet manipulation and graphics, it is possible to have more immediate interpretive access to factual information by decision-makers.

This is an example of the way in which a technological development fits with a management style. Accordingly, it becomes integrated into the behavior of the organization.

·As other territory is brought into focus, constructs such as Decision Support Systems, Matrix Organization, and Zero-Based Budgeting may use the micro as a vehicle. In each case, integration occurs.

Each individual who proposes alternatives will be supporting them with readily available data. The spreadsheet and graphics, then are a fit for other non-technology dynamics that are occurring in the organization.

The idea that there is a fit does not mean unchallenged harmony. Integration is not tranquility, but is growth errupting out of foment that makes sense by the rationale of the model.

Likewise, there will be such problems as reliability of statistics from remote files down-loaded from a data base. But the point here is that the support that is marshalled for an alternative is based upon prototyping that an individual has done at a keyboard with a financial model.

4.2 A Cost Incentive for the Natural Cultivation of a New Generation of Computer-literate Middle Managers

Another illustration of integration may be management computer literacy. For years automation proponents have looked for the day when end-users would be qualified to see systems from the perspective of ADP components. To some this did not seem to be imminent until a generation of managers is replaced by a generation of graduates who studied business courses which included ADP. Even if suitable on-the-job training were available, how could that much training occur for an entire generation of managers?

It now appears that such will happen as a grass roots process.

Where economics spurs the acquisition of micros, users will be prompted to become technically functional with them. Whether they will universally be exposed to good system development procedures or not, they will at least achieve a conversant level of functionality. Individual training will be demanded.

Hence a compatible goal will be achieved almost as a spin-off, or as an itch that gets scratched.

4.3 Emergence of the "pro-user"

Dissipation of the Wedge Which had been Driven Between the ADP Technician and the End-user

Ever since the ADP professional became aware that users exist and began to call them "end-users" there has been a polarization between them. It may be time for this distinction to evaporate. Would this not be an example of integration? Similarly as Alvin Toffler sees the roles of the consumer and producer merging into the "pro-sumer", perhaps we also are seeing the emergence of the "pro-user."

A "pro-user" may be anyone who uses computer products to process and access the information he wants.

The previously labeled "end-user" may become more skilled with his new tools than is the ADP professional. Intensity of computer related skills may no longer be a role distinction between these groups.

Everyone will be managing information for organization goals. And everyone will have to play keep-up with the technology, regardless of whose turf he is functioning in.

This can be a new treatment for provincialism in the organization and a case of integration.

4.4 Re-defining Information

In response to the question, "What is information?" I'm sure you are developing some new answers. How we are enriching our view of what information is speaks clearly of integration. At one level it emphasizes the global scope of concerns. At another level it speaks of the character of the organization which is defined by its information.

Far from the early ADP view of data as transactions only, information now comes from the PC, the terminal, the word processor, the mailbox and the office cluster.

Information can be as follows:

Corporate data
Local ADP data
Research data
Processed data
Extracted data
Administrative data
Decision Support data
Word Processing data
Operant data

The data may not be different. Its usage is different and has meaning throughout the organization--and when one makes interpretations that run throughout the organization, that suggests integration.

This may sound like the archaic M.I.S. promises. The difference is that M.I.S. presumed that one could define in one mega-plan a super definition of all data in the organization. The dialectical/construct model does not assume that static view. It works with process. The micro allows us to re-define information. Such re-definition can be at the service of the organization. Accordingly, integration comes by applying new constructs to old data. Thus, integration is based upon flux; not upon static definition. The process of zooming, construct application, and goal orientation encourage that flux to support the organization's productivity and profitability.

In addition to its global nature, the new information reflects the character of the organization. All of industry, government, economics, civilization and culture depend upon the symbols of information. What makes the differences among the Super Bowl, the Mardi Gras, a high school commencement, the Miss America Pageant and the Democratic National Convention? Isn't a lot of it, what data is taken to mean and how it is processed?

As more job functions in the organization become knowledge-defined, and information-constituted, the character of the organization becomes more and more dependent upon those information functions.

Similarly as the DNA code is a constituent component of the make-up of an organism, information is a constituent component of the organization. In that sense, the organization is information.

So, the way in which information is manipulated by the many end-users impacts integration of the organization. This manipulation is supportive for the organization insofar as it is directed by the dialectical model dynamics.

We have noted that the dialectic can employ integration in the example areas of management methods, literacy, the pro-user and re-definition of information.

The tools used for accomplishing those integrating processes are the "constructs." I am now going to discuss two examples of constructs from current organization experience.

5. Examples of Constructs

I have mentioned that constructs can be reference systems, beliefs, laws, principles, perspectives, criteria, environmental frameworks, pictures and models.

The two case study examples have all these elements, but can most clearly be considered environmental frameworks with reference system under-pinnings.

The construct serves as a tool for the dialectic modeling process. The construct provides the ground rules so that participants know what game they are playing, so that actions have continuity, and so that inappropriate actions are not taken.

5.1 The Micro-lab

A construct I would like to suggest as a case study is the use of a microcomputer laboratory. Sometimes a construct is a stated philosophy. In this case, the lab is not a philosophy, but it represents a philosophy. Basically it is the philosophy which treats the end-user like an adolescent who is going to college. Rather than controlling him, you give him some new tools and trust his background to carry him on from there. The lab itself is an "environmental framework" in the sense that it is an organizational entity. Its existence contributes to the execution of the dialectic model. At the present time, at the State of Missouri, we are in the first months of establishing such a laboratory.

The context for the lab is an environment in which numerous divisions will be considering acquisition and application of microcomputers. The division sponsoring the lab offers to be of service to the many user divisions. The sponsor division also is interested in the overall health of ADP enterprises in the user divisions. As far as authority is concerned, the user divisions are essentially autonomous with regard to microcomputers.

Micro proliferation is seen as both an opportunity and a possible danger. Its main strengths are its weaknesses. Its rapid payback, responsiveness, accessibility to "non-professionals", freedom from constraints, and piece-meal decision items suggest to many observers the need for coherent multi-division leadership and planning mechanisms.

The dialectical construct model suggests the micro laboratory as an approach.

First of all, the model suggests that we ask the right question. Some might be tempted to have asked, "Who is right, who is wrong?" or "How does an organization control an elusively acquired resource?" or "What resource type is cost beneficial?"

Rather than those kinds of questions, the model would suggest the question, "How do we contribute to organizational growth by addressing integration and tension factors?" The lab encourages access to the tools needed to accomplish integration like the four forms of integration that we discussed in the last section (4.1 - 4.4).

This is done by giving end-users an opportunity for hands-on experience, education, assistance, prototyping, comparing, project magnitude assessment, demonstration and experimentation.

The primary service is access to operational resources, both hardware and software. The posture of the lab is to adapt to user functions. What one user cultivates is to be shared with others. So, the climate arises out of practice. Likewise, skills that are found to be needed are cultivated and addressed with training aids. A variety of hardware and software is maintained to offer the user the primary options that others in the organization are finding helpful.

It is intended that divisions will benefit in their own system development projects, planning and acquisition of products.

Exposure to the lab may or may not lead to uniformity of some kind across the multiple divisions. The philosophy is that users are self-directing, but should have advantage of the experience of others and exposure of major resources that others are finding useful.

How was the micro lab suggested by the dialectical construct model?

On the surface it may have seemed that we had a technological or an economic issue. In fact, it was an issue of organization dynamics and the solution is an organization development solution.

The lab reflects the service posture of the sponsor division. This is part of the environmental framework and reflects an organizational viewpoint.

It is not a territorial power kind of move. It is a contribution to a global learning situation.

The lab will be used where the need arises. Thus, in response to tension, its use becomes a part of the dialectical cycle.

Organizational learning is the result. And organizational learning is an experience of people and agencies who modify their approaches as they discover new results.

5.2 TEAM Development of Organizational Information Management Services

Another construct example comes from an experiment with "teamwork."

The context for this construct has been the need for definition of organizational functions, services and projects. Its most major use was during a time of reorganization transition. Further use will be primarily for brainstorming and joint planning in new project areas.

Pre-conditions

The reorganization transition was a phase during which several staff members were in new positions, the division director was new, and partial reorganization was around the corner. The "Team" involved eight (8) people in a planning and coordinating section. This section is part of a division of 140 employees. The division, in turn, performs services for departments with several thousand employees. The scope of the Team effort was primarily within its own area of responsibility. The nature of that, however, extended concern into matters of division level functions.

The Team's section had responsibility areas which were relatively easy to adjust by management discretion. The new director had conceptual interests and skill which were adaptable to the organizational dialogue which was emerging. The staff was vocal. The director was ready to hear and to interact.

The staff had interest in discussion about goals and approaches. The new director had interest in reviewing current practice and forging new directions.

Approach

A staff member suggested a one-day brainstorming session at a remote site in a retreat setting. The suggestion was accepted and eight (8) people met spanning three (3) levels of supervision.

The agenda included presentations and dialogue about what was being done and what was recommended.

Near the conclusion the director delivered the gauntlet, saying in effect, "You are a team, carry on from here in your own way--and there is something I want as output. I want your recommendation as to the criteria for the services this division should be offering."

This was, in effect, an invitation to start at ground-zero for what the section should be doing.

The Team began weekly meetings with rotating chairmanship. Throughout its functioning there were expressions from some indicating desire to be told more about what was expected. Others encouraged moving ahead. The director insisted on keeping hands-off.

Results

The weekly meetings were held for four months, with substantial individual and subcommittee work being done between meetings. Formal documentation was developed. Recommendations were made of functions the division should be performing. Reorganization occurred. New major directions were established.

Under the reorganized structure some members were uncomfortable with continuing Team activity. Other members wanted to continue.

Management decision was to comprise a continuing Team of the new supervisors. The total staff would function as a team on an "as needed" basis.

Relationship to the Model

The dialectic application of constructs was experienced throughout the Team experience. There were repeated cycles of zooming for tension and for integration. An interesting thing about that is that what the director was wanting to see was process. He wanted to see those three levels of folks hammering out ideas in dialogue. And that is a natural for the dialectical model.

I will mention here some tensions leading to the Team idea, some tensions resulting and what the contribution was to the division.

Tensions Leading to use of the Construct

A. Lack of sense of identity by individuals with any stated objective of the division.

B. A variety of unstructured and undefined viewpoints about what the division was doing or should be doing.

C. Uncertainty about roles, expectations, and responsibilities.

Tensions Resulting from Use of the Construct

A. Where expectations of some members of the Team (depending upon their perception of objectives) became fixed, they felt a lack of follow-through.

B. Where the Team idea reflected a participative management style, sometimes staff liked being included; sometimes staff did not like the responsibility, the loss of targets for complaints and the fear of being penalized for expressing views disfavored by supervisors present.

C. Where authoritarian management style moves re-asserted themselves; sometimes staff were relieved at being led again and at having problems solved by someone else; sometimes disliked being told what to do when left out of the decision process.

Contribution of the Experience to the Division

The team experience has been an example of the state of flux that often characterizes the dialectical process. There has been a lot of give and take; a lot of ambiguity; and some solid ideas that benefitted from the scrutiny they had received. And the team approach exhibits the thematic, methodological traits of a construct. There has been organizational learning and growth in the sense of staff working together in a new context. Clarity of work has been evolving.

There has been progress toward goals. Emerging out of the teamwork, management has made thematic and procedural moves toward re-definition of what the division does.

The idea of the Micro-lab came out of the Team process.

The microcomputer laboratory and the team process provide interpretive frameworks in a fluid situation. They contribute structure that is not static. Such is necessary for constructs in the dialectical process.

6. Wrap-up

As microcomputers are infused into the organization, parochial interests run head-on into global concerns.

The microcomputer has appeal as it is made responsive to organizational tensions. Its infusion results in new tensions. Application of constructs may bring integration or may surface new tensions if equilibrium needs attention. Constructs like the lab and the team can work with the tension cycle for growth and learning in the organization. Results like fact-based management, computer literacy, and the pro-user may reflect integration within the cycle.

The model advocates fluid processes. It depends upon change. Likewise it should be applied to embrionic areas such as integrated application packages on the micro and to end-user application development.

SESSION OVERVIEW
FEDERAL MICROCOMPUTER ACTIVITIES

Allen L. Hankinson

National Bureau of Standards
Washington, D.C. 20234

Microcomputer-based systems represent a new frontier in information technology. These systems offer the potential for Federal agencies and other information intensive organizations to improve substantively the productivity of the workforce as well as the quality of the services being provided.

These organizations face a stiff challenge to realize the potential offered by this new technology. A major problem is that the technology is moving so fast that it is difficult to get a handle on it's effective management and use.

Some of the critical issues that must be addressed include:

- when should these systems be considered as alternatives and/or adjuncts to central facilities;
- what are the key steps in the process that should be used in the selection of these systems;
- how should these systems be configured to enhance the sharing of data and the sharing of expensive peripherals;
- how should applications software be acquired in order to minimize development and maintenance costs;
- how should this technology be "packaged" to meet the needs of nontechnical end-users.

In this session, Federal microcomputer activities that involve these and other issues will be explored from three different perspectives:

- from the perspective of an organization which develops Government-wide acquisition programs;
- from the perspective of an organization which develops Government-wide technical standards and guidelines;
- from the perspective of an organization which has already taken some innovative steps to deal with these issues within a Federal agency.

DATA PROCESSING USER SERVICE - A PROBLEM; A PROPOSED SOLUTION

Thomas H. Acklen

Veterans Administration Data Processing Center
1615 East Woodward Street
Austin, Texas 78772

While data processing technology continues to progress at an ever increasing pace, techniques employed by data processing organizations to extend these technological benefits to users remains basically static. Users are buffered by analysts and programmers from the equipment's capabilities. Maintenance of existing systems places a growing burden on the data processing organization, and the development of new applications is people-intensive and protracted. Users are becoming more dissatisfied with the data processing organization's inability to respond to their demands. As a result, users are in many instances, trying to use microcomputers to fulfill their own needs. Such maverick efforts, although potentially beneficial to a particular user, could introduce disarray into an organization's efforts to establish integrated information systems.

This paper proposes a series of actions which should improve the data processing organization's ability to serve the user community. The underlying strategy of these actions is to enable users, within the context of an overall data processing plan, to provide for many of their own needs thus permitting the data processing organization to concentrate on the more complex tasks. The overall objective is to insure more responsive and less costly data processing support.

Key words: Communications networks; data manipulation capabilities; data repositories; programming productivity aids; responsiveness; software improvement plan; systems development methodology.

1. Introduction

During the late 1830's and early 1840's artist/inventor, Samuel Morse, perfected the telegraph and devised a standard code for transmitting messages electronically. And thus began the communication revolution and the modern day need to improve user service.

The user that wished to send a telegraph message had two available options. Under the first option, he relayed his message to a telegraph operator and requested that he code and send the message to the desired destination. The operator at the receiving station decoded the message and relayed it to the specified individual. If all worked well, the telegraph operator(s):

(1) understood the user's message,

(2) properly coded it,

(3) routed it to the proper destination,

(4) correctly decoded it, and

(5) delivered it to the intended person.

Under the second option, the user could master this technological wizardry that required him to speak in dots and dashes, find an operator that would allow him access to a telegraph key and send his own message. Neither approach could be candidly described as "user friendly".

With the invention of the telephone, Alexander Graham Bell radically altered the electronic communication user interface and the method and type of support provided the user. The user became the direct conveyor of the message. Nevertheless, in the early days of the telephone, responsibility for routing of the message still rested with the equipment operator. Today, the user of a modern telephone system both routes and conveys the message.

Although there are exceptions in today's data processing world, user support is about equivalent to that which existed in the early days of the telegraph. In order to benefit from the computer's capabilities, the user must either depend upon data processing technicians or become a data processing technician. Inherent in such an environment are costly delays, poor service, and dissatisfied users.

The challenge for today's data processing manager is to place the analogous telephone in the user's hand. First, he must through a combination of tools, techniques, training and trust, eliminate when possible the analysts and programmers that buffer the user from the computer's capabilities, thereby allowing the user to directly utilize this technology. Second, he must strive to improve the traditional user/analyst relationship and the support provided the user through this interface. This paper proposes a series of actions designed to assist the data processing manager in meeting both objectives.

2. The Problem

Our industries and government institutions can ill afford the inefficiencies associated with today's multilayered, people-intense methods of data processing systems development and maintenance. As the unit cost for computer hardware has plunged, the labor cost associated with application software development and maintenance has skyrocketed to the point where it accounts for the vast majority of an organization's data processing budget. Even if these institutions were willing to commit the requisite dollars, our labor market would be hard-pressed to provide the number of analysts and programmers required to meet rapidly expanding data processing demands. The U.S. Department of Labor forecasts that between 1980 and 1990 the need for programmers will increase by about 40% while the demand for systems analysts will climb by 50%. It holds that such rapid increases in the demand curve will cause a continued increase in labor costs and an overall decline in the availability of data processing talent and skills.

Our institutions face the additional threat that frustrated users, aided by the ubiquitous microcomputer, will reinject into the data processing environment (1) nonstandardized design techniques, (2) duplicative development efforts, and (3) parochial attitudes regarding the gathering and sharing of information. These are reminiscent of the vices that the data processing world has struggled to free itself from for the past ten years. This is not to imply that such unilateral efforts may not benefit the particular user. However, the nonstandardized and undisciplined methods of these "computer illiterates" may, in many ways, do our institutions a disservice. Our ability to effectively plan and integrate data processing services for our organizations, and to devise and implement systems that share corporate data, will be impeded by these maverick systems.

3. The Traditional Solution

Reasons for change in our current approach to users support are readily apparent; however, the avenues to such change are far less clear. In selecting an avenue, it is extremely important to understand the particulars of the service provided and the specific problems which a user has with that service. An often-voiced problem with today's data processing service is lack of responsiveness. Changes needed by the user yesterday cannot be provided for six months or longer if, because of other requirements with higher priority, they can be provided at all. Another frequent complaint is that the changes, once finally made, are not what the user really wanted or needed in the first place.

Traditionally, we as data processing professionals have attempted to address these problems by using the approach which we best understand, "Give me <u>more</u>--more analysts to better evaluate what the user needs and more programmers to more quickly code programs that meet these needs." We work more closely with the user in the early systems design stages to better understand the changes that he desires. But this technique has yielded basically unsatisfactory results because its underlying premise is that the user knows what he needs to better do his job when in many cases he does not. We reason that by establishing larger programmer staffs we may (1) more rapidly maintain aged and often poorly documented systems and (2) write more programs to meet the new demands. However, constant maintenance of old code becomes increasingly difficult and the results less acceptable to the user and the demand for new code outstrips our ability to write it. In summary, we use more resources to do more of those things which have proved marginally successful at best.

4. A Possible Alternative

Given that the traditional approach to user support has proved inadequate, another which is more responsive to and more supportive of the user must be formulated. Such an approach should embrace all phases of the application systems life cycle from identification of a need through the design, implementation, maintenance, and finally redesign of a system that fulfills that need. This approach should:

1. Stress the maintenance of corporate data repositories which are readily accessible to authorized users.

2. Feature a data communications network which allows for the effective gathering and dissemination of corporate data. This network should be organization/function oriented rather than application oriented.

3. Afford the user simplified yet powerful data manipulation capabilities.

4. Emphasize the self-service approach to data processing. It should assist and guide the user so that he may fulfill many of his own data processing requirements within the context of a company-wide data processing system.

5. Recognize those tasks which are best performed by the data processing shop, and in those cases, use analytical tools such as demonstration screens and simple data manipulation capabilities to help both the data processing shop and the user in better understanding what the user requires.

6. Feature a "minute man" team of top-notch technicians that can rapidly respond to unforeseen, priority user needs.

7. Employ the latest programming productivity aids to make that application programming which is done by the data processing organization less costly and less time-consuming.

8. Incorporate a master software improvement plan designed to gradually free the data processing organization from the costly and time-consuming burden of maintaining poorly documented and obsolete application programs.

9. Utilize a highly disciplined system development methodology for all new applications to insure that they are more standardized, maintainable, and responsive to the user.

The philosophy underlying this approach is a recognition that the data processing organization cannot effectively or efficiently respond to all user demands. Its objectives are to (1) maintain an integrated data processing and information sharing capability within the organization, (2) allow the users to assume responsibility for some of their data processing requirements, and (3) permit the data processing professional to more effectively accomplish those tasks which cannot be readily assumed by the user.

The approach de-emphasizes application programming by the data processing professional. It is well to note that at the state-of-the-art will not allow the elimination of this fundamental task. As we progress beyond somewhat primitive, natural language programming capabilities toward artificial intelligence and voice recognition capabilities, the need for data processing professionals to create application programs will diminish. However, this approach concentrates on that which is achievable today. A more detailed examination of each component of the approach may provide insight as to how it may be adapted to your organization's particular needs.

4.1 Provide Ready Access to Corporate Data

Data, regardless of why or by whom it was initially gathered, is potentially valuable to other organizational elements. Within the context of data processing, information has traditionally been viewed as the property of the gatherer and stored and accessed in a manner that primarily benefited that individual or organizational element. The owner (gatherer) of that data often resisted its use by other organizational elements.

This perception must change. The concept that data is a corporate resource to be made available to all authorized users for the overall benefit of the organization is fundamental to effective information management. This concept can best be achieved by creating a data administration function which reports to upper management, preferably the executive officer of the organization. In the area of information management, both the data processing shop and the organization's business elements, should be subordinate to the data administrator. This will insure that all users are better served.

The data administrator function can only remove the organizational impediments to data access. The technical impediments can best be addressed by designing application functions around a database management system. A modern commercial database management system will permit your organization to share data and manage information. Such packages afford far greater flexibility in the storage, sharing, and maintenance of data than was possible under the flat file concept of data storage. Organization standards and policies will be required to insure that data is stored in such a manner that it can be easily retrieved and manipulated by authorized users. Without such standards your database management system will be only a sophisticated data storage and retrieval tool. By establishing and enforcing standards the data processing shop will ensure the existence of an information repository capable of serving many corporate users.

4.2 Implement an Information Communications Network

To be useful data must be gathered, manipulated, and/or disseminated. Traditionally, we have designed gathering and dissemination capabilities each time we developed the data manipulation capabilities of a particular application. However, in order to be responsive to users, it is far more effective to establish a common data communication network between the data processing shop and those remote locations where data originates or is disseminated. Do not expend resources developing a network for payroll data or sales data or inventory management. Instead develop an information communications network. Use a modular design that allows for expansion. Select a widely used network protocol that can support various types of terminals and processors. Such a common data link will facilitate rapid development of more effective user applications.

We, as data processing technicians, often look at an information system and compare the merits of centralized versus distributed processing. We must also begin to think in terms of centralized versus distributed development. We must not forget that a well-planned information communications network facilitates both.

4.3 Provide Users Simplified Data Manipulation Capabilities

Under this approach users must be provided software packages which allow them to easily and economically retrieve and manipulate data. Most commercial database packages afford such capabilities. Software packages such as SAS, SAS GRAPHICS, DYL-280, SPSS, are also designed to be used by the data processing novice. Furthermore, new product offerings of this nature are being introduced almost daily. Consequently, it is neither prudent nor economical to have your systems analysts attempt to understand and plan an application for those tasks that users, if given the proper tools, can provide for themselves.

4.4 Support the Concept of Self-Service

Simply acquiring such capabilities will not insure a more satisfied user community. Your center's capabilities must be marketed to the users. Additionally, key people within your organization must be convinced that allowing users to provide for their own needs is sound data processing management.

Establishment of an Information Technology Center to introduce users to (1) your center's capabilities, (2) equipment which can be used to access your center, and (3) the basic skills required to benefit from the support your center offers is an effective mechanism for marketing the center's capabilities. Demonstrating the capabilities of microcomputers in accomplishing small tasks will aid in familiarizing users with data processing. Also demonstrating the shortcomings of small computers in rapidly accessing and manipulating large volumes of data is important in marketing the concept of fully integrated systems. Teaching users how, if certain standards and conventions are followed, microcomputers can work in unison with your center is mandatory if the concept is to be achieved.

Once users are sold on the idea of self-service, your organization must be prepared to offer ongoing support if they are to be satisfied. A mobile team of skilled technicians that can, upon request, visit a users shop to aid them in their tasks will be required. Your shop will have to guarantee prompt and reliable service during the hours users require it, i.e., guaranteed service levels.

Gaining user interest may be easier than gaining key data processing personnel support for the self-service approach. Because of legitimate concerns for the service provided and a lingering concern for job security, many managers within the data processing shop may oppose allowing users access to data and data manipulation capabilities. Their opinions must either be changed or nullified; the former being preferable to the latter. Through example and exposure these managers must learn that (1) users can effectively master basic automatic data processing skills and accomplish data processing tasks on their own, and (2) just as eliminating the need for a telegraph key operator did not eliminate jobs in the communications industry, so will allowing the users to aid themselves not threaten the job security of the data processing professional.

4.5 Employ Progressive Analytical Techniques

Although extending computing capabilities to the users will do much to satisfy their data processing demands, it will by no means replace the requirement that your organization develop the more complex automated systems that support your organization. In effectively performing this function, it is critical that the data processing professional understand what the user wants and needs; that is, ensure that the system designed and implemented is the system that will meet the organization's needs. As previously discussed, this can be a very illusive goal since users generally are not fully aware of their needs nor of how such needs can best be fulfilled through the use of computers.

Computer applications contain three basic functions (1) data gathering, (2) data manipulation, and (3) information dissemination. Failure of the latter function equates to failure of the system in fulfilling a particular business requirement. Thorough understanding of user requirements with regard to the latter function will dictate an effective system design. Systems analysts using terminals, sample data, and variable screen formats can learn, along with users, what information is needed to fulfill each business function. This approach allows the user to see how the application output will look, evaluate whether it meets the organization's needs, and if not, suggest and shortly thereafter review proposed changes to the output. This approach will avoid much retrofitting often required in the latter stages of a system design. It will also more effectively guide the analysts in determining what data must be gathered and how must it be compiled and manipulated. The final result will be a system which satisfies organizational requirements.

4.6 "Minute Man" Response for Unanticipated Tasks

Proper planning is critical in order to provide effective service to data processing users. However, data processing tasks will arise which were simply unanticipated. Such unanticipated tasks may be time sensitive and critical to the successful functioning of your organization. Forcing such tasks into the routine program development/maintenance workflow is disruptive and often counterproductive.

If an unanticipated task is both time sensitive and critical to the organization's function, it should be referred to a "minute man" team for execution and implementation. Even though it represents a substantive investment, the team nucleus, composed of an aggressive manager and two or three top-rated technicians, should be a permanent organizational element. Depending on the nature of the task, this team should be able to draw selected talent from other organizational elements. Upon completion of a particular task, maintenance of the resulting system should become the responsibility of the cognizant line organization, borrowed talent should return to their respective organizations, and the team nucleus should begin preparing for the next task.

Benefits of this approach are:

(1) the assembled team will appreciate the urgency of the task,

(2) the talents applied to the project are those required to rapidly accomplish it,

(3) disruptions to other projects are minimized, and

(4) the needs of the organization are more rapidly served.

But remember the task assigned to such a team should be <u>unanticipated</u>, <u>time sensitive</u>, and <u>critical</u>. Use of the team must be the exception, not the rule.

4.7 Rely on Productivity Enhancing Tools

Productivity aids can be used to reduce the time and cost of developing and documenting systems. Such tools range from natural language programming packages to programmer work stations that support interactive compiling, editing and debugging. Use of these tools can, given the particular task, more than double programmer productivity.

A clearing house function should be established within your shop to review all user requests that require programming to determine whether code-generating tools can simplify the task. Once such a determination is made the task should be assigned to the cognizant project manager. The clearing house function removes this decision from the project manager who, because of familiarity, may elect standard coding techniques over a more advantageous code-generating package. Whenever possible, management's objective in this area should be to supplant people-produced code with that generated by machine.

4.8 Aggressively Pursue A Software Improvement Plan

If your data processing shop is representative of the industry norm, 70 to 80 percent of your programmers' and analysts' time is spent maintaining existing code, consequently leaving few resources for new efforts. Furthermore, your organization can afford neither the dollars nor the time required to completely redesign/replace those systems that require such heavy maintenance. Because of large capital investments in old application code, your organization is in a "darned if you do; darned if you don't" dilemma.

A well thought-out software improvement plan will allow you to salvage a large portion of your capital investment while reducing overall maintenance costs. Application code can be improved without a complete system redesign. In order to accomplish this, each application must be analyzed to determine what modules account for the greatest portion of maintenance costs. These modules should be targeted for major modification or replacement. Care should be taken in avoiding the logic that machine language code is costly to maintain. If such code is basically static, it costs little or nothing to maintain. The governing rule should be rework those modules/programs that require frequent and time consuming maintenance. And during each effort, exercise care not to produce more code which will be difficult to maintain.

The point is not to suggest how to improve your particular software. The point is to suggest that it can be--that it must be improved through an evolutionary process. The first step is most difficult; establishing a software improvement project, and committing the organization to specific software improvement goals. Management must insist that all system changes be accomplished within the context of this plan. Long-term objectives accompanied by discipline, perseverance, and accountability will result in more maintainable software, more efficient code, less dependence on a particular individual's knowledge, and more rapid response to user needs.

4.9 Utilize a Highly Disciplined Systems Design Methodology

The problems of poorly documented, aged, and patched programs that require more effort to maintain and modify than was required to initially develop, are not the result of a devious conspiracy by previous managers to undermine the data processing operation. To the contrary, these problems have resulted because dedicated managers and technicians, in the name of expediency and user support, employed unorthodox techniques and shortcuts. In the development phase, they did more with less, so today we do less with more.

In recognition of the problem, companies have developed and now market highly structured approaches to govern and guide development efforts. Although these approaches differ in specifics and degree of automation, they are common in the following aspects:

a. A life cycle step-by-step approach to systems development is prescribed.

b. User responsibility/participation in the requirement definitions, design alternatives, and functional specifications is mandated.

c. Form and content of the technical specifications are defined.

d. User and data processing management review and concurrence at predefined intervals are specified.

e. Documentation standards are precise and unyielding.

These approaches provide a simple but rigid skeleton which if built upon will support the muscles and organs of a complex system. Systems developed using these approaches are better documented and less complex than those designed using less structured techniques. Consequently, these systems should be less costly to maintain and modify.

New systems development projects that are the responsibility of the data processing organization should be accomplished within the confines of one of these structured approaches. The organization selecting a highly disciplined system design methodology, committing to the overhead associated with its use, and holding managers accountable for enforcing its provisions, will not have to spend its tomorrows trying to undo yesterday's expediencies.

5. The Costs of Change

Adaptation and implementation of this approach to user support may require numerous policy, organizational, and staffing changes. Resources typically devoted to the programmer and analysts functions will have to be diverted to communications network support, database design and maintenance, marketing and training and one other topic not yet discussed--capacity management and resource acquisition. The latter is mentioned because this approach will require substantial amounts of computer hardware and software. The approach requires the application of computer power to data processing tasks in order to make them more efficient. Data processing managers thought little of using computer capacity to automate payroll or inventory management functions. They must be equally bold in automating their own efforts. The result will be a more satisfied and better served user, a more effective organization, and a more efficient data processing shop. Remember, there is more to data management and communication than tapping a telegraph key.

STANDARD COSTING FOR ADP SERVICES

David R. Vincent

Institute for Software Engineering
510 Oakmead Parkway
Sunnyvale, CA 94086

This paper is an initial exploration into the area of standard costing for ADP services. Historically, data processing expenses have been regarded as variable, depending on hardware and software procurement and usage. This paper takes the position that information systems expenses are relatively fixed, and that information is the resource to be managed, as opposed to hardware and/or software.

Key words: Accounting methodologies; ADP services; data processing; data transfer; information resource characteristics; management of the database; standard costing; storage of the information asset.

The information systems community has proliferated numerous "accounting" methodologies, techniques and even packaged software products that have been designed to calculate and report the "true" ADP/EDP/IS unit, standard, and/or chargeback basis for the costs of operations. The sad fact is that none of these methodologies, techniques, or methodologies reflects the real nature of information systems costs the way that they exist in today's environment, and, more importantly, the way in which we will view information systems costs when we begin to consider that it is information that is the resource as opposed to just the hardware and software used to make that information available to the user.

Moreover, I/S expenses tend to be relatively fixed when we consider them from a month-to-month basis. This is the way in which top management tends to view the I/S investment, and it is one that they wish to minimize in order to live within the fiscal limitations imposed by the annual funding cycle. The irony comes with cost reports that show unit costs and chargeback amounts in government environments where they must "charge users with the cost of their usage." With these kinds of reports and "MIS-information" comes the lower organization concept that information systems costs are variable. Armed with this often dangerous information, users will tend to minimize their use of the "EDP resource," many times by taking their work to outside service bureaus, which causes the I/S costs to the rest of the users (and to the organization as a whole) to go up! Certainly, the people who drafted OMB Circular A-121 didn't have this in mind when it was formulated.

Traditional standard costing principles were developed in the manufacturing environment to account for variable material, labor, and overhead on a unit of manufacture basis. In the factory situation, material is not used unless there are units to produce, and labor is not called in or kept on the job unless there are units to produce. Most of the overhead goes on, so there may be an overhead absorption problem, but variable costs may be inventoried or avoided simply by not purchasing. Similarly, traditional governmental accounting has always been aimed at a somewhat variable personnel and materials expense budget.

In the information systems environment, the truly variable costs are few, such as paper, reels of tape, electricity, and any temporary portion of labor. All the rest of the costs are relatively fixed such as hardware, software, development, staff, facilities, and management. Even with all the good intentions of GSA, an increasing fixed investment in software will not allow the hardware expenditures to become variable.

The only real variable involved is the work that can be processed by these relatively fixed costs, which is analogous to distribution of manufacturing overheads. A more appropriate method for understanding and analyzing information systems costs is, therefore, greatly needed. We also need to redefine what we are trying to analyze as the information systems environment changes from focusing on a centralized, fixed-cost/overhead, support function to analyzing the investment, cost of creating, and cost of maintaining the organizational resource information.

The first step toward resolving this issue lies in understanding that information systems is only a small component of a much larger information resource. The primary sorts of information resource activity that will fall under the aegis of what is currently called the Director of ADP (and is likely to be called the Chief Information Officer in the future) may be described by the following four major areas:

- Storage of the information asset
- Management of the database
- Data processing
- Data transfer

1. Storage of the Information Asset

With the growth of the organizational information asset, the storage of that resource takes on new significance. It is the author's opinion that by the end of this decade, the accounting community will insist on reporting the information asset as a fixed asset of the organization just like any other asset (e.g., cash, equipment, buildings, inventory) and that this asset will represent the single largest asset for most governmental organizations. This, coupled with the advent of fixed disks on direct access storage devices that consume more and more physical space, causes this asset to represent a larger and larger investment to the organization. Other types of information storage exist such as tape and mass storage devices, but with the increasing use of information on a real-time basis, their appropriateness will diminish (one possible exception is an emerging technology involving the use of optical discs that may be removable, storable, and shippable, and are rumored to cost less than tape). The appropriateness of data to be stored at all, as well as the decision as to which method of storage is appropriate, will be resolved only by analyzing the costs of storing information as well as the costs of database management for the organization requiring information storage.

THIS PAPER ASSUMES THAT TOP MANAGEMENT AND THE USERS OF INFORMATION ARE THE ONLY GROUP THAT CAN ASSIGN VALUE TO THE INFORMATION RESOURCE. IT IS UP TO INFORMATION SYSTEMS MANAGEMENT TO PROVIDE THE USERS AND TOP MANAGEMENT WITH SUFFICIENT COST INFORMATION TO ENSURE THAT THEY WILL BE ABLE TO MAKE INFORMED COST AND INVESTMENT DECISIONS, ESPECIALLY THOSE REGARDING TRADITIONAL EDP RESOURCES SUCH AS CPUSAND THE LIKE (this was the real intention of OMB Circular A-21).

2. Management of the Database

Database management includes all the hardware and software necessary to make that information available for retrieval. The security of the data and data integrity are ensured by the adequate management of the information. This will be the most difficult area to manage in the future. Visionaries in this area such as James Martin, Bill Synnott, and Bill Inmon, for example, spend most of their time trying to develop long-range thinking regarding these demands. For those of you concerned with sub-second response time, this area will provide the greatest challenge. What is seen today with systems such as IMS, IDMS, etc. can only provide an inkling of what is to come. We are already hearing much about relational and distributed vs. centralized databases.

3. Data Processing

Users have traditionally used the central processing facility for their processing needs. With the advent of minicomputers, some began to establish their own processing resource, while continuing to utilize the central processor as well. With the increasing proliferation of personal computers, it will become even more popular to process one's own information locally, especially with the heightened requirement of response time. Therefore, it will be necessary to assist the users in the tradeoff decision as to whether they will use centralized or distributed processing resources by providing them with central processing cost data. This can be compared with their local processing costs as well as the advantages of sub-second response time. However, there may still be costs for information storage and management from a central facility. These costs should be analyzed from the tradeoffs concerning local versus remote storage (and what the corporate policy is regarding this) as well as the costs to ship the information where it will be used, whether that be over a telecommunications network, by express or regular mail, or whatever. This cost category is covered in the next segment regarding the transfer of data.

4. Data Transfer

In the case of a centralized data base, or even in the case of distributed data bases, there is the need for data transfer when there are distributed terminals or processors. This cost is essentially a communications cost and will become extremely important as file transfer becomes more popular due to the use of personal computers to do local processing. The author feels that this will be the major information cost of the future (even though the advent of optical discs may provide some temporary relief).

The characteristic behavior of the above areas of information is unique. The methods of measuring, costing, and analyzing each area will require separate considerations such as local versus remote processing as well as information storage, data management and relevant communications (data transfer) trade-offs. Table 1 sets forth the salient characteristics for each area to be used as a guideline. A metric for each is stated, but this does not mean that the measurements from the four different areas of activity may be added to express an overall measure of capacity as has been expressed by such concepts as software physics, computer resource units, MVS service units, and the like. Adding the four areas together would obfuscate the unique characteristics of each. Even though clever people can think up some kinds of algorithms to explain system behavior with combined statistics, top management and the users of information will not be able to understand the algorithms or be able to make easily the important trade-off decisions re the I/S investment. This kind of reporting would be analagous to the utilization reports put out by many I/S departments showing such things as CPU utilization in percentages. What kinds of decisions in today's environment can be made on that kind of information?

The costs for storage of information will be associated with the type of media used and the space occupied. The alternate metric of bytes may be the way that space is defined as opposed to tracks, tapes, etc. For the purposes of this discussion, it really doesn't matter. What does matter is how to relate the investment made in storage to how it is used to satisfy users' information needs.

Database management costs should include all the I/S resources needed to get information into and out of storage. The kinds of resources used will include CPUs, disk and tape devices, controllers, and various software and operating subsystems. The important thing is that the database management costs will be associated with the time that an I/O system is used, the amount of time that a device is used, or -- with the advent of cache memory devices -- the time that cache is used. This would also apply to any other media such as mass storage or tape devices. These costs (not to be confused with storage costs) are for the retrieval of information as opposed to the cost of storing the data.

The data processing cost, as mentioned before, is directly associated with the central processor and its operating systems, personnel, and other overheads. It may also be associated with the distributed processors, which are very easy to cost since they are fully located in user departments. This is especially true in governmental organizations, whose procurement policy makes it much easier to purchase micro - or minicomputers within an organization.

The transfer costs are comparable to telephone costs and must represent the amount of time that a communication link is used. This link should also include the distance characteristic, because communications costs go up proportionately with distance.

In summary, the use of this kind of revised analytical approach to management and costing will enhance the effort by making it understandable to both users and management. It also makes it possible to relate the costs to the burgeoning organizational resource known as information and the trade-offs that exist in the procurement of ADP resources.

Table 1. Information resource characteristics

Metric	Storage	Management	Processing	Transfer
Basic	space	time	time	time/ distance
Alternate	bytes	bytes	bytes	bytes/ distance
Other	media	system	peak/ time	peak/ time

AUTOMATING CONFIGURATION MANAGEMENT

Enrique G. DeJesus, 1Lt USAF
Craig J. Riesberg, 1Lt USAF

HQ MAC/ADCI
Scott AFB, Ill. 62225

The ability to manage Software and Hardware Changes at remote sites is one of the most critical components in any computer network. Four major elements of configuration management are addressed: Change Control, Validation Testing, Inventory Management and Software Distribution.

The first element, Change Control, provides a complete audit trail (document tracking) of change requests from conception to implementation, regardless of origin (self or user initiated). The control includes, but is not limited to, automatic numbering, cataloging incomplete requests and journalizing historical data upon completion.

The second element, Validation Testing, provides for internal software driven flexible benchmark type testing; or an external remote terminal emulator which again uses the flexible benchmark concept but adds stress testing capability.

The third element, Inventory Management, deals with the particular applicability of the change to the site taking into consideration the hardware configuration.

The fourth element, Software Distribution, handles automatic shipment of bundled software to sites configured on line. Further, it provides automatic preparation for shipment via the most expeditious means available to sites not accessable through the computer network.

Key-words: Change control; inventory management; software distribution; validation testing.

1. Introduction

Most federal agencies and civilian organizations rely heavily on their computer networks to accomplish daily operations. With the current impetus toward distributed processing, changing bundled software at remote sites becomes an ever increasing problem. Not only is it necessary to transfer new software to the geographically separated locations, it is of utmost importance that the software perform as expected. In today's rapid-paced data processing environment, operational systems rely on timely and adequately tested software. This paper discusses a system which: 1. allows users to request changes and enhancements and at the same time provides the host organization the means to control those requests (change control), 2. tests the newly-created software prior to field implementation (validation testing), 3. allows the host organization to control the software and hardware inventory in the field (configuration management) and 4. provides various means for the distribution of bundled software (software distribution). The system is modularized so that any organization can implement any module independent of any other.

2. System Overview

The entire process of changing software at remote sites can generally be broken down into eleven distinct and separate steps (see figure 1). In order to provide reliable software these steps must be followed in precise order. While several of the operations are outside the scope of the automated distribution system the operations are enumerated to give the total view. Each step is listed along with its place in our system.

1. Identification of Requirements - Performed by the user or system builder who identifies a new requirement or the need for an enhancement (outside of system).

2. Control of Requested Changes - Performed by the software control agency, possibly with their order of importance (change control).

3. Evaluation of Request - Must be an independent organization or individual to remain objective as to cost versus expected gain (outside of system).

4. Analyzing and Coding New Software - Performed by host organization personnel or contracted to a software agency (outside of system).

5. Development Testing - Conducted by a group supporting the writers. May only test the new software in a stand-alone mode without testing its interaction with other systems (outside of system).

6. Documentation of New Software - Performed by writers (outside of system).

7. Verification of Documentation - Performed by the software control agency. Determines if new software changes are supported in the user's/operator's manual (outside of system).

8. Software Bundling - Performed by the software control agency. Will use source programs to provide bundled software for various types of machines (inventory management).

9. Release Testing - Performed by the software control agency. Will test the interaction of new software with other systems by a benchmark or through the use of a remote terminal emulator (validation testing).

10. Distribution - Performed by the software control agency. Forwards the bundled software to the sites via available and reasonable means (software distribution).

11. Historical Documentation - Performed by system maintainer. Shows new changes and modifications to the existing system. A continuation of item number 2 (change control).

3. Change Control

Software changes can be triggered by new user's requirements or problems encountered during normal logic execution. In either case, it is one of the forces that can affect the operational integrity of any computer system. These changes are usually organized in documents which serve as the official voice of the new requirements or modifications needed. These enhancements must be approved by the software control agency before they can be passed to the appropriate working agencies.

It is extremely critical that the software control agency maintain an audit trail of the software enhancements being worked or already worked within the system or systems under its control. By the tracking of these documents, the software developer is kept informed of enhancements or changes that need to be accomplished in order to meet the user's corporate requirements. On the other hand, the document originator receives two levels of acknowledgement. First, the software control agency approves the request and passes it to the software developer. Second, after the requirement or fix has been satisfied by the developer, the result is a software release to the controlling agency with the appropriate documentation. This release is then passed back to the document originator. The change control module of our system will allow for: 1. the automatic numbering of documents, 2. transmission and acknowledgement, and 3. historical data management (see figure 2).

Once a software enhancement has been identified the user will have the capability to perform an online software change request. The automatic numbering module will accept the change request assigning a unique number. By this number the system will be able to determine the document originator, system or subsystem affected, software developer, priority and the agency which will receive the new release. As soon as the document is accepted it will be passed to the transmission and acknowledgement module. This module is responsible for the electronic transfer of all active change requests to the respective software developer's working agencies. In addition, upon arrival of the software release package from the software developer, this module will update the change request database in accordance with the documentation provided in that package. The historical data management module will provide housekeeping reports of change requests overdue or outstanding for any period of time. Furthermore, it can supply the administration

ELEVEN STEPS IN SOFTWARE CHANGE

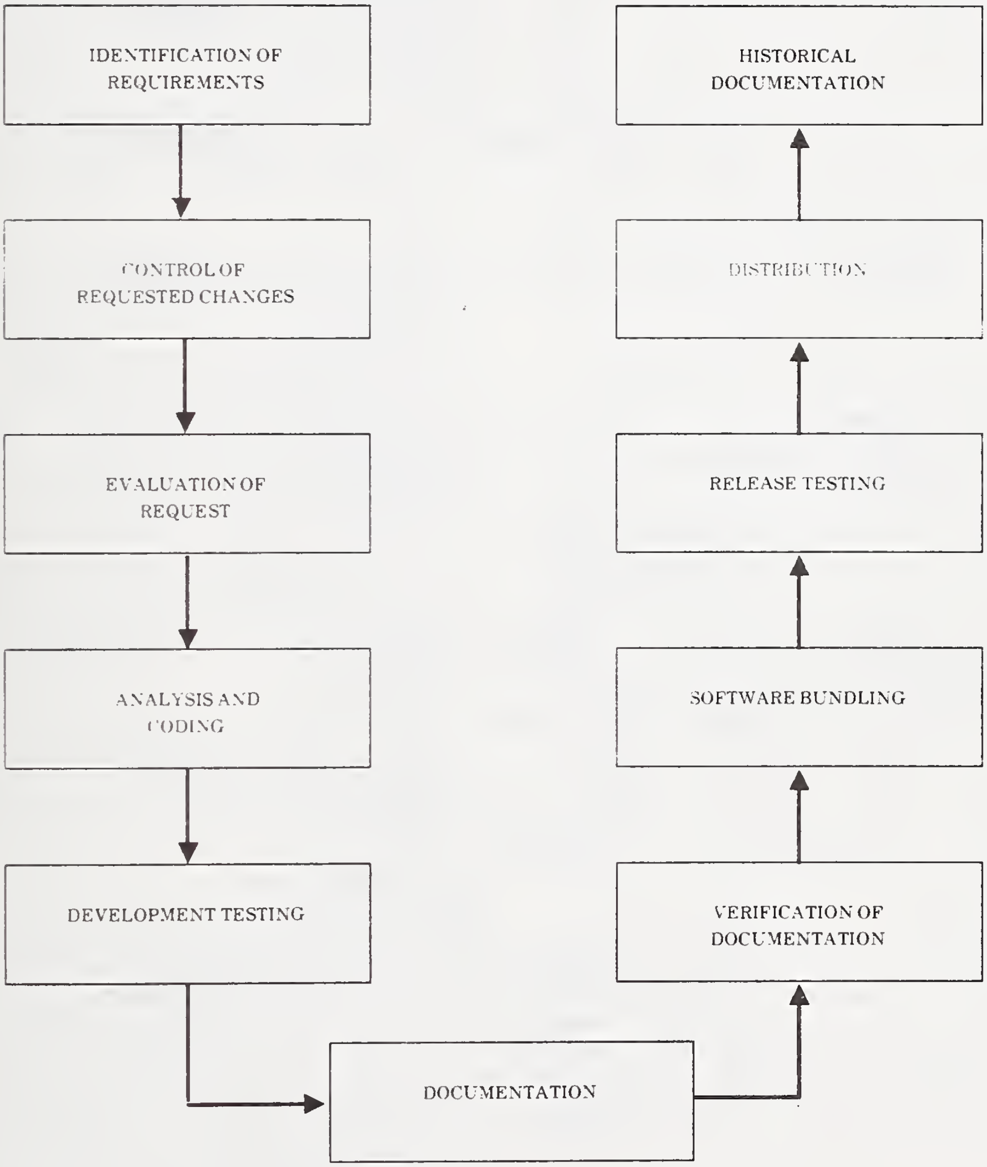


Figure 1. Steps in Software Change

CHANGE CONTROL FUNCTION

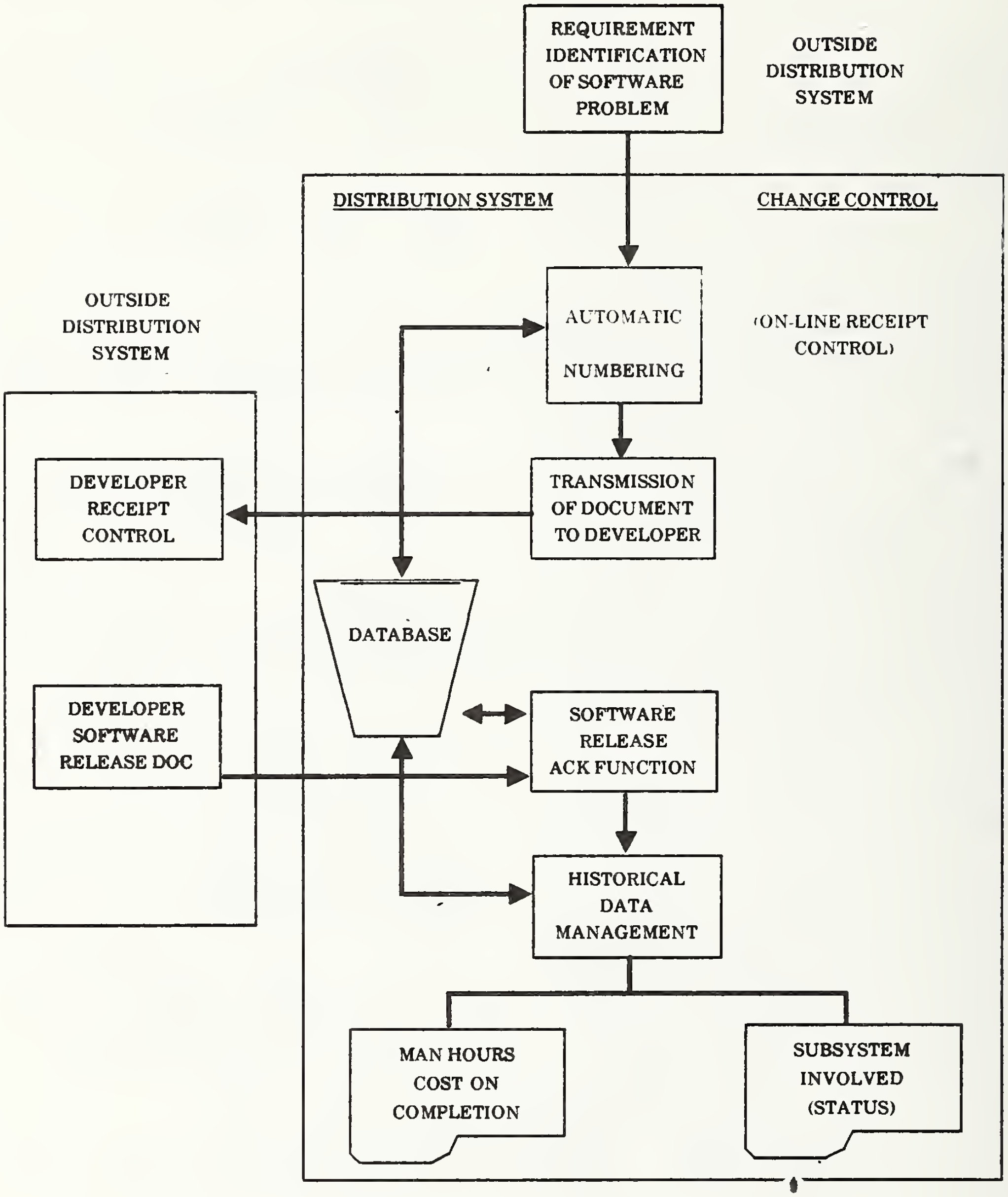

REQUIREMENT IDENTIFICATION OF SOFTWARE PROBLEM
OUTSIDE DISTRIBUTION SYSTEM
DISTRIBUTION SYSTEM
CHANGE CONTROL
AUTOMATIC NUMBERING
(ON-LINE RECEIPT CONTROL)
OUTSIDE DISTRIBUTION SYSTEM
DEVELOPER RECEIPT CONTROL
TRANSMISSION OF DOCUMENT TO DEVELOPER
DATABASE
DEVELOPER SOFTWARE RELEASE DOC
SOFTWARE RELEASE ACK FUNCTION
HISTORICAL DATA MANAGEMENT
MAN HOURS COST ON COMPLETION
SUBSYSTEM INVOLVED (STATUS)

Figure 2. Change Control

with total or partial cost of software enhancement or system development.

4. Validation Testing

The Validation Testing module of our distribution system is designed to exercise and verify the newly created software against the system or systems affected.

Exhaustive testing of the software to be released is the key item for a successful implementation on the computer network. Our proposed system will: 1. build and maintain test scenario files for all systems specified, 2. perform external or internal system tests, 3. report to the test director using online/report (hard copy) modes of the test findings and 4. evaluate the performance of the software during the preceding test (see figure 3).

In order to effectively test a software release, we need to organize a baseline of all the transactions allowed in the system or systems in question. This baseline must identify the transactions in three major areas: software function performed, input data and expected output. The first function on the validation testing module allows the creation of a transaction driven test scenario with the capabilities of performing online additions, modifications or deletions of transactions already established. The accuracy of the contents of such scenarios will be the critical factor for a valid test evaluation and a successful implementation of the released software on the network.

Another element that needs to be taken into consideration when building the scenario file(s) is the system test database(s). Cost versus reliability might be a factor when determining the size and contents of the test database.

Once the scenarios are built, they are executed in the second function of the validation testing module. The execution of the test scenarios might be external or internal to the system(s) under test depending upon the host organization resources. The external driven test will use a Remote Terminal Emulator (RTE) as the test driver. The RTE will be loaded on hardware outside the system(s) under test environment, eliminating the competition for system resources once the test has been started. Some advantages in using the external test mode are the network operational simulation of inputs through the physical communication lines allowed in the system(s) under test and the ability to perform stress test not only on the system under test but on the network itself. A disadvantage might be the cost of the additional hardware used by the RTE to drive the system(s) under test.

The internal driven test uses a software test driver to validate the release. The software driver shares the system(s) under test resources and passes the transactions specified on the test scenario file(s) in a single threaded mode to the appropriate application's software.

Both testing methods mentioned above, external and internal, will validate each and every output received from the system(s) under test against the baseline output specified by the test controlling agency in the system test scenario file(s). Any mismatches are reported to the online test director's device established at the beginning of the test. This is the primary role of the third function of the validation testing module. The test director will have to make a decision concerning the test. Possible alternatives are: 1. continue the test, 2. retry the transaction that caused the mismatch, and 3. restart or stop the entire system test. Particular attention should be given to the system under test's database(s) when deciding further actions after a transaction mismatch is detected.

The successful execution of the system(s) test scenario will provide high confidence in the operational reliability of the released software once loaded on the network. Furthermore, after the test of the operational capability, our validation testing module will spawn a Computer Performance Evaluation (CPE) package as the last validation function. During the execution of the system(s) scenario test file, CPE data can be collected and stored in different medias (e.g. tape or disk) so that elapsed times for different functions (transactions) can be analyzed and compared with previous CPE results. Service times (the time it takes application software to work a requested function) as well as Response times (total elapsed time from user's terminal to application's software and return) can be analyzed so that unexpected results can be evaluated prior to release.

Other levels of validation might be performed in addition to the four functions described. One such validation might be performed on the documentation provided by the software developer in the software release package. For example: the system operator's manual can be inspected to insure that new error messages or system commands are included in the package.

5. Inventory Management

One of the largest problems in business management is knowing what goes where and who gets what. Management of computer resources is no exception. The Inventory Management section of our system is designed to provide a complete inventory, both hardware and software, of each

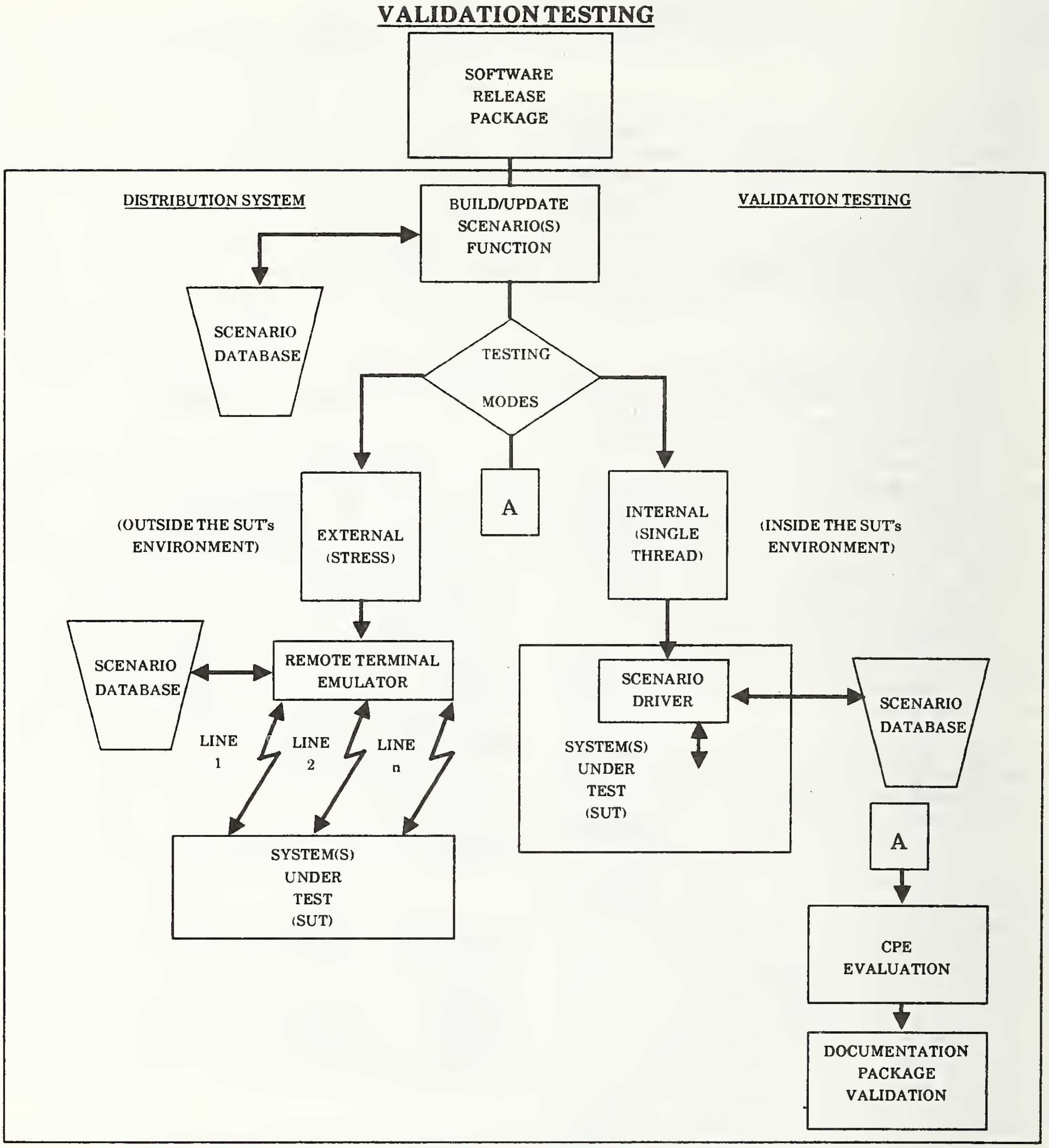


Figure 3. Validation Testing

individual remote site. By utilizing the information gathered and stored by the inventory management module, the system is able to determine compiling and bundling requirements for various types of machines. This information is also shared with the distribution module during the shipping phase of a software release. Since it is information which is shared, the inventory management module allows for the storage of information to be used by other modules (e.g. site addresses for the distribution module which will be used to prepare mailing labels). The inventory is broken into two distinct divisions; hardware and software (see figure 4).

Effective hardware inventory management can provide help in at least three different areas. These three areas; system documentation, cost analysis and capacity or upgrade planning are intricately interwoven in the inventory management module. Each area can be developed as needed beginning with system documentation. System documentation provides the basic information needed about any remote site. It provides a list of what is currently available at the site in regards to CPU, memory, work stations and peripherals connected. By combining system documentation along with purchase, lease and maintenance contract costs the inventory management module can provide cost analysis data for any given site in any given configuration. After including more information concerning the given systems maximum capacities regarding memory and peripherals, it can provide a type of capacity planning as well as cost factors to support an upgrade to a higher capacity.

Software inventory management, while providing report generation capabilities, is used primarily by other modules of the distribution system. The validation testing module uses the software inventory to determine which configurations need to be tested when new software is released. By maintaining a list of bundled software for each site, only those configurations which will ultimately receive the new software will be tested. It is also through this module that the releasing organization can accurately estimate the amount of time necessary to bundle, test and distribute new software.

The software distribution module utilizes the software inventory in conjunction with the hardware inventory to determine which sites will receive the new software releases after it is successfully tested. As stated earlier, it not only determines which software is to be released but which mode of transmission will be used to transport the software. The inventory management module maintains site addresses for postal and electronic mail.

By examining hardware and software inventory at sites containing more than one machine of the same make, it may be possible to reconfigure one of those machines so that two different systems could be run on it in a degraded mode. This hybrid backup system would be extremely useful in keeping an essential system operational during a period of machine failure. It could also be used to provide service during the relocation of a system.

6. Software Distribution

The software distribution module of our system is responsible for the automatic shipment of all new software after it has been bundled and tested. It insures that all remote sites in the network receive the version of the new software that is compatible with their hardware. The actual distribution of the software lends itself well to automation because it is generally repetitive work. Unique features of the distribution module include the following capabilities: 1. distribution to an operational test site prior to general distribution, 2. use of different modes of distribution based on the urgency and size of the released software, 3. inclusion of instructions on how to load and operate the new software and 4. automatic acknowledgement (see figure 5).

The first feature, the ability to distribute to an operational test site, is almost an extension of the validation testing module. In the validation testing module the new software has been rigorously validated against a test scenario. However, certain aspects of the new software, such as overflowing storage tables and indexes can only be exercised in an operational mode. Distributing the new software to an operational test site provides this additional mode of testing prior to its general release. If different sites execute different portions of the software, multiple sites can be chosen and controlled by the distribution module. After a specified period of time, such as 30 days, if there are no problems with the software it will be automatically released to all applicable sites in the network. An additional benefit of the operational test site is that it allows for the validation of loading instructions and user's/operator's manual by field personnel prior to general release.

The second feature is the capability to use different modes of distribution based on the urgency or size of the release. Determining the mode of distribution will generally require an analysis of need versus cost. Generally, the fastest method of distribution will be the most costly. Available means of distribution include but are not limited to; physical distribution of tapes or disks and distribution via electronic mail or direct communication links.

Most releases will be of a routine nature and can be accommodated by a relatively simple distribution system. In its most elementary state the system will copy the new software to a

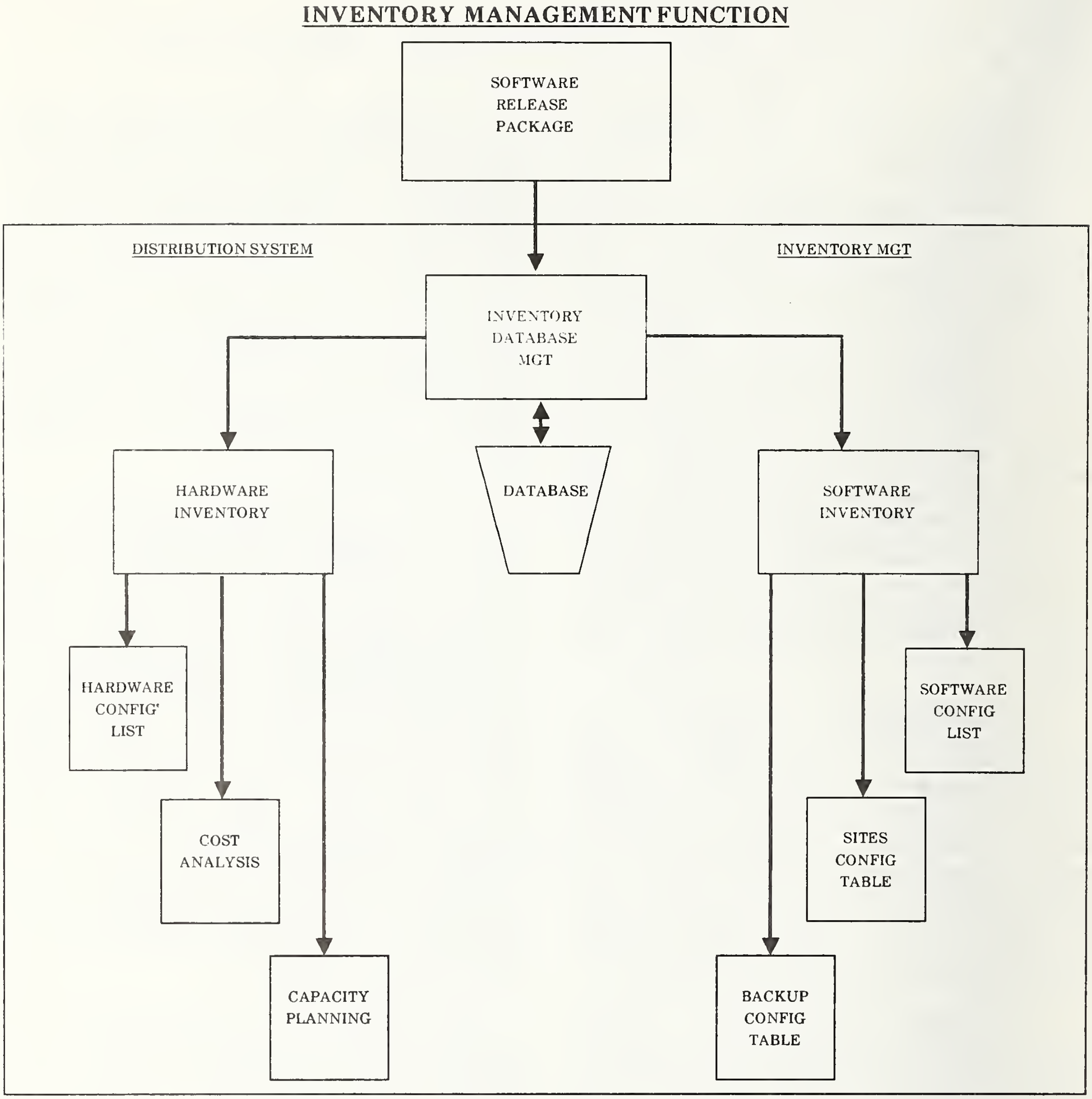


Figure 4. Inventory Management

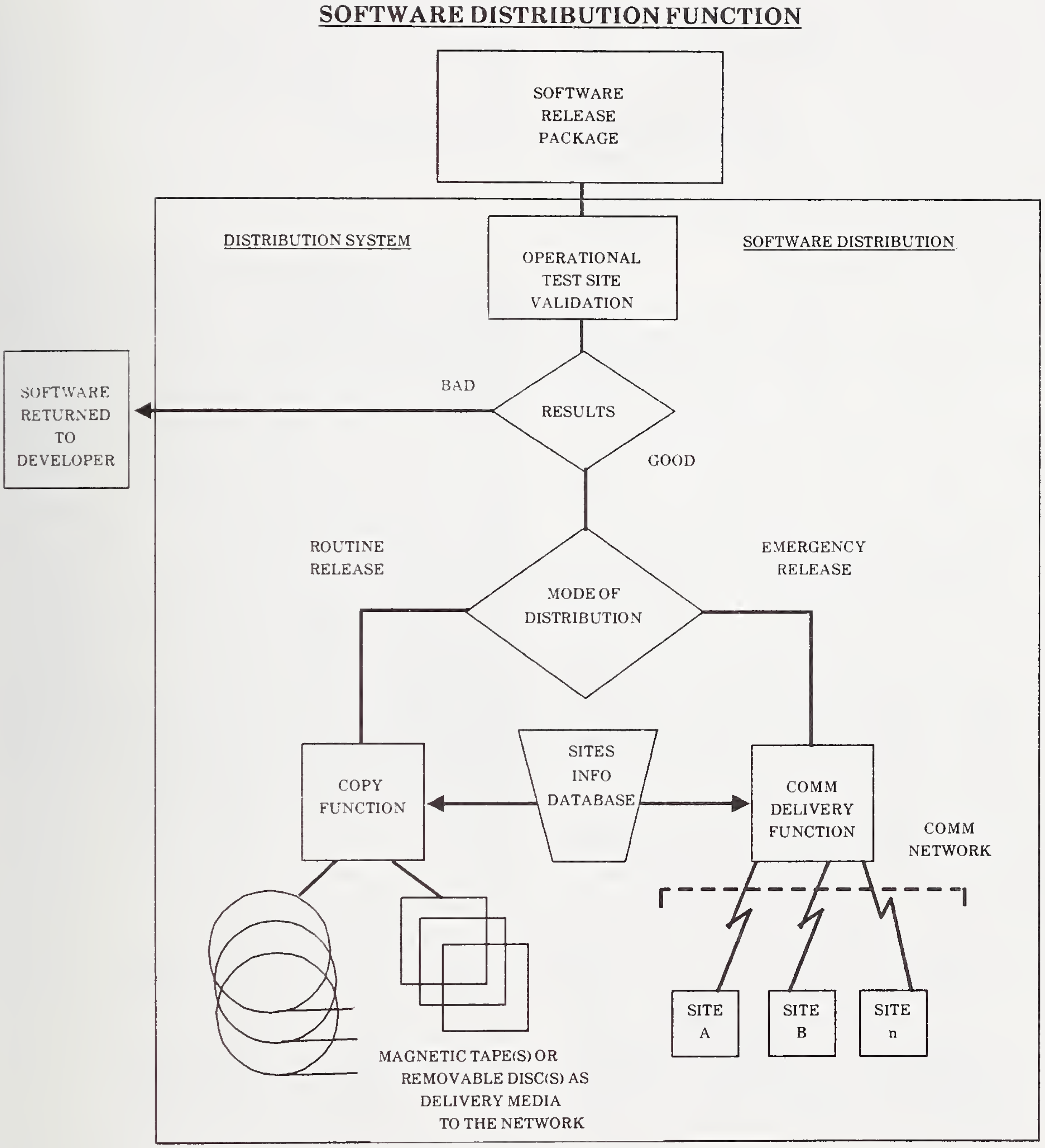


Figure 5. Software Distribution

magnetic tape or disk, providing one copy for each site that is to be a recipient of the new software. It will then print a mailing label and a set of loading instructions for each site. The main advantage of this method is that it can handle the largest releases, including completely new operating systems. Additional advantages are that it minimizes shipping costs and the updated user's/operator's manual can accompany the software release package. The main disadvantage is that it is slow in moving software to the remote sites. Part of this can be attributed to the need to generate one copy for each site, and partly to the available means of transportation to remote sites, especially outside the continental United States. The copy and forward method can also be used to transfer hybrid backup systems (see inventory management) to be used during system failure.

The second type of distribution is for time critical emergency releases. This type of distribution utilizes the existing communication links of the system or some special link to quickly transfer the new software to a particular site. Other factors to be considered when selecting this method include the size of the release, the number of sites which must receive it and the availability of the communciation link for an extended period of time. The main advantage of this method is its speed. Emergency patches to software can be quickly transfered along with the loading instructions. Another advantage is the use of the networks own communication system to transmit a single copy to all remote sites concurrently. The main disadvantage is the high cost to transfer over long distances to numerous sites. Other disadvantages are that the network itself must be operational and that a large release could tie up the communications link for an extended period of time.

The next feature is the software distribution module's capability to incorporate instructions into the release package. While the receiving site may have the knowledge to load its own software, this seemingly insignificant capability allows the sender to furnish specific instructions for each site. The loading instructions can have a significant impact on the software's usefulness. For example, new software might be time sensative - simultaneous loading of new communication software may be required at all sites - loading at any other time may cause edit errors and incorrect updates.

The final feature to be discussed is automatic acknowledgement. Two types of return acknowledgement should be built into any system. The first is acknowledge-ment of receipt of the new software. If the distribution was through the network's communication links this can be automatic; otherwise, the receiving site must manually trigger a return message stating when the new software was received. It is important for the releasing agency to have a historical record of when and how each release was received by the remote sites. The second acknowledgement is generated by the remote site. It is an informational message telling the distributing office that the remote site has complied with the instructions and is or is not currently using the new software. If the remote site encountered any problems in loading or operating the new release they may also include narrative description of the problem in this message. This return information is stored by the software distribution module for a specified period of time.

7. Concluding Remarks

Any large organization deeply involved in distributed processing would benefit from an automated configuration management system. After determining the need, but realizing that the resources are limited, the next question is logically "Where should we start?". Because operational sites are so dependent upon the host for valid software and because the organization itself may not survive without online data processing, the validation testing section is our choice for initial development. It assures the distribution system that the software being released is worth the effort; however, it is hard to distribute the software without knowing who needs it. For this reason we chose the inventory management module as the second step. The inventory management information must be in a format readily available for software distribution; therefore, it is natural for these two units to be developed concurrently, leaving change control as our last effort. Regardless of order of development, each module should be as independent as possible to allow for future enhancement and redesign.

The Terminal Probe Method Revisited. Some Statistical Considerations.

Luis Felipe Cabrera

Departamento de Ciencia de la Computacion
Escuela de Ingenieria
Pontificia Universidad Catolica de Chile
Casilla 114-D
Santiago, Chile

Abstract

The Terminal Probe method has recently been used to compare selected performance indices of different interactive computer installations. Since these comparisons have been done under the "natural" work load of the systems several validity questions arise.

In this paper we present results which summarize our experience in 1Ø systems with more than 11.ØØØ measurements made over a span of two years. We analyze the empirical behavior of response time and find system independent statistical properties which enable us to attain predetermined confidence intervals for our values. Moreover, we have found a family of statistical models which fit our data in a comprehensive way: not only the mean and variance of our response time distributions are well approximated but the modelled distributions fit the observed ones. Thus, ordered statistics can also be obtained.

We may now control the data gathering period better, and, what is more, we have a predetermined statistical confidence in our curves. Thus, the robustness of the method is justified for comparisons.

Key words: Benchmarking; generalized linear models, installation comparisons; linear predictor; performance; performance indices; terminal probe; UNIX operating system; work load estimators.

1.Ø INTRODUCTION

Over the years the so called Terminal Probe Method has been utilized in a variety of contexts. Indeed, not only it has been used to assess a

----
This research was supported in part by the National Science Foundation grant MCS-8Ø129ØØ and by a Pontificia Universidad Catolica de Chile research grant DIUC 47/82.

specific configuration under a controlled work load, but there have also been studies which use it as a tool for comparing different installations operating under their "natural" work loads. Thus, questions regarding representativity and validity of these comparisons have been posed. These questions have remained unanswered.

In this paper we will present a study which shows that there exist system independent statistical properties of performance data, gathered using the Terminal Probe method, which enable us to draw firm conclusions from studies based on approximate work load estimators. Moreover, the study also shows how one can limit the data gathering period and how one may select some predetermined values of the work load estimators as sole objects of measurement, thus reducing the method's overhead even more.

Our results are based on 11,813 measuremnts taken on 10 systems over a span of two years. All of the systems were presented with the same portable benchmark which runs on any UNIX system. Even though our benchmark gave us three "time" measurements; user, system and response time, we have limited our discussion to response time.

Section 2 briefly explains the data gathering method and the systems studied. Section 3 gives a study of the correlations observed between our work load estimators. Section 4 summarizes our study of the response time distributions, as functions of our work load estimators. Section 5 introduces our main statistical tool, the generalized linear models, and presents the approximations obtained with GLIM. Finally, we present our conclusions in Section 6.

2.0 THE DATA GATHERING METHOD AND THE SYSTEMS MEASURED

All of the installations used for our measurements run versions of UNIX operating systems. UNIX is a trademark for a family of operating systems developed at Bell Laboratories over the last 14 years [11,14]. In 1969 a first version was implemented on a PDP-7, and since then they were ported to PDP-11s, VAXes, IBMs, Amdahls, Interdatas, NCRs and others. In 1979, a group at the University of California at Berkeley implemented a paged virtual memory extension to UNIX for the VAX [1], which is now part of what is known as "Berkeley UNIX". Most of our measurements were done in this last type of system.

On logging in in a UNIX system each user is assigned a special process containing a command interpreter, known as the "shell" [3], which listens to the terminal. The shell parses the input line and decodes the command requested along with its flags and arguments. Then it execs the command. Shells may also read commands from files. Thus users may define sequences of shell commands, known as shell scripts, and store them in files for later invocation. The versatility of these scripts is greatly enhanced by the fact that the shell language also contains control-flow primitives, string-valued variables and arithmetic facilities. Since UNIX automatically handles all file allocation decisions, the portability of these scripts is greatly facilitated.

The strategy for monitoring responsiveness of each system was the same as that used in [4]. It consists of running a script which has a set of predefined benchmarks together with commands which gather statistics about the work load and measure the time it takes the benchmark to terminate. The script runs periodically in a totally automatic way. Each time the script cycles through its commands, it executes a "sleep" command that suspends its execution and then wakes it up after a predetermined number of seconds. We decided to use the time command because of our commitment to use standard UNIX tools. Time has a rather low resolution and truncates, it does not round off. More details can be found in [5,6].

This data gathering method can be categorized as a time-sampling method [9] and is in fact very similar to Karush's terminal probe method [10]. By using it with a system in normal operation one evaluates the performance of an installation.

Table 1 presents a summary of the number of measurements for the various systems. Hardware changes were made to an installation in some cases. For statistical reasons we had to consider the systems which existed before and after the change as being different. (For example, in Table 1 VAA and VAB correspond to two such systems. This change, a fairly trivial one, was the addition of some ports to the installation.) The most important

System	No. of Measurements
VI	3427
P7A	3206
VM	1467
P7B	1016
VAA	844
VC	791
VB	435
P4A	301
P4B	235
VAB	91
Total :	11,813

Table 1: Systems measured and the number of observations.

mnemonic element in the names is given by the first letter. Those names which begin with a P are PDP-11 systems. Those with a V are VAX systems. All VAXes are 11/780, [7]. If a 7 appears it means 11/70 and a 4 means 11/40.

There were a total of four different installations, three at the University of California, Berkeley, and one at Purdue University. Those measurements labeled VC come from Purdue, where the VAX 11/780 had, at that time, a configuration with 3M bytes of main memory, 56 ports, three RM03 disk drives on Massbus 0 and one TE 16 tape drive on Massbus 1. Measurements from P4A and P4B came from a PDP 11/40 which had 200K bytes of main memory, one DIVA disk controller and three DIVA disk drives with 50M bytes disks. This installation had 23 ports and no floating point arithmetic unit. The P4B measurements were made on the installation without a cache memory, and the P4A with a 2K cache memory. The P7B measurements were made on a PDP 11/70 with 1.3M bytes of main memory, 2K cache memory, one DIVA disk controller with four DIVA disk drives and an RS04 fixed head disk used as swapping device. This installation had 81 ports. The P7A measurements were made on the same installation with 2M bytes of main memory and where the drum was used for storing temporary files instead of as a swapping device.

All of the other measurements were made at the same installation which went through sucessive changes. VB was an 11/780 with 512K of main memory, two RP06 disk drives, one TE 16 tape drive and 16 ports. This was a "swapping" UNIX system. VAB was the same installation but running a paging version of UNIX. VAA was VAB with 8 more ports installed. VM had 2M bytes of main memory, 2 RP06 disk drives, two CDC 300M byte disk drives and 32 ports. VI had 4M bytes of main memory, 4 RP06 disk drives, two CDC 300M byte disk drives and 72 ports.

3.0 CORRELATIONS AMONG WORK LOAD ESTIMATORS

Throughout our data gathering period we ran a script which contained three basic tasks: a C compilation, a CPU bound job and a text formating job. We also measured three work load estimators: the number of users logged in (nu), the number of processes in the process table (np), and the number of active users (nau). Nau was obtained by counting the number of ports which had more than one process associated with them. This generated a bias in systems where there were several deamons running, but no correction was made because the same number existed during the entire data gathering period. We also measured the global response time of this mix, which we called script.

At a latter stage, and only for the systems VI and P7A, we appended two new tasks to exercise aspects of the systems which were found not to be well individualized. These tasks did not alter the measurements taken of the initial mix. They were a copy of a 60K byte file within the same disk, and an editing session involving a series of commands made to a 60K byte file.

The choice of these portable work load estimators has been documented [4,5,6]. Nevertheless, their correlations were now studied. As they are related to each other, it was of interest to decide if we could explain the same phenomena with just a subset of them. Table 2 presents the results obtained for the correlation coefficients corresponding to all the measurements in each system.

From Table 2 we observe that there exists substantial correlation between the different pairs of estimators. These coefficients are certainly larger than those observed in the social sciences, which are on the order of .4,

System	Correlation Coefficients		
	nu vs nau	nau vs np	nu vs np
VI	.899	.799	.689
P7A	.917	.849	.791
VM	.880	.854	.783
P7B	.946	.903	.884
VAA	.852	.838	.746
VC	.881	.769	.686
VB	.842	.855	.771
P4A	.677	.828	.660
P4B	.410	.345	.470
VAB	.492	.906	.606

Table 2: Correlation coefficients between work load estimators for the different systems.

and smaller than those obtained in economics, which are on the order of .95. Moreover, we also see that the pair which tends to be less correlated is nu vs np, which suggests that these two estimators measure different aspects of the work load. We also observe that for P4A and VAB, nau and np are much more highly correlated than the other two pairs. In Section 5 we shall see that for most systems and tasks, nau adds no significant additional information to that given by nu and np.

Using the SPSS facility called SCATTERGRAM [13], we were able to plot response time for our different tasks as an individual function of each of our work load estimators. Moreover, scattergram also gives us an idea as to how each cross section distribution looks. It is unfortunate that it prints values only through the digit 9, but further analysis enabled us to plot the exact histogram, and hence the observed shape of each distribution. We shall discuss this in Section 4.

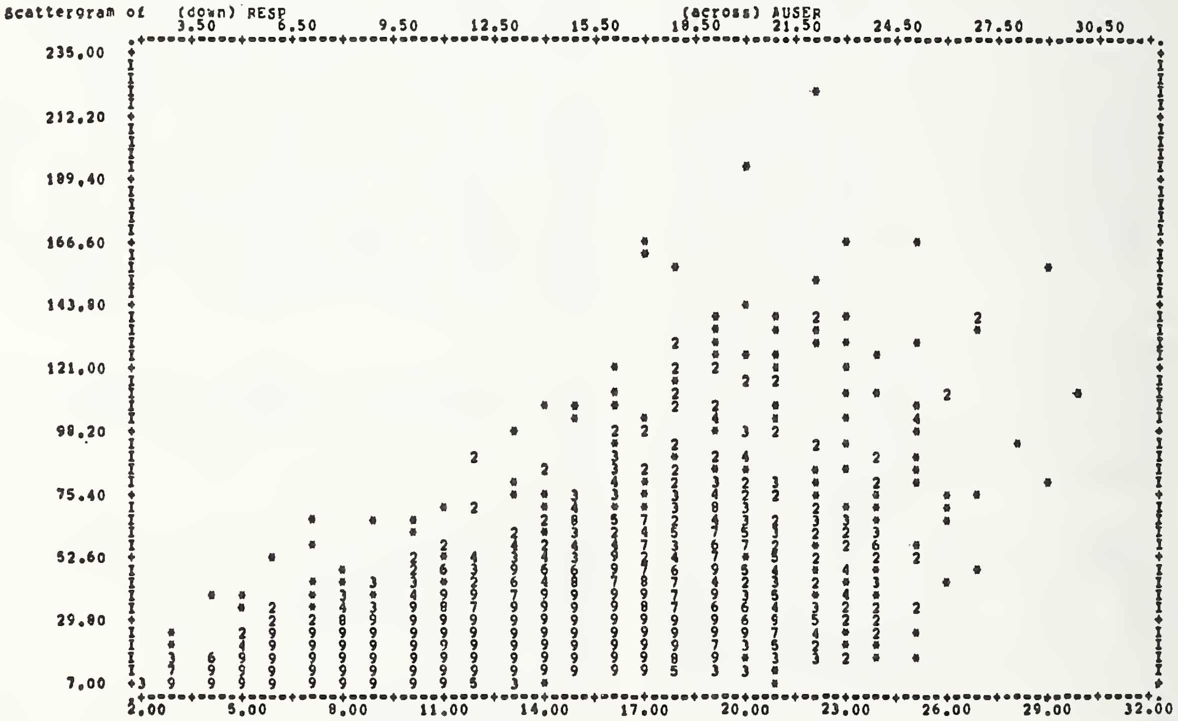


Figure 1: System VI. Scattergram of the task man man. Nau (AUSER) versus rt (RESP).

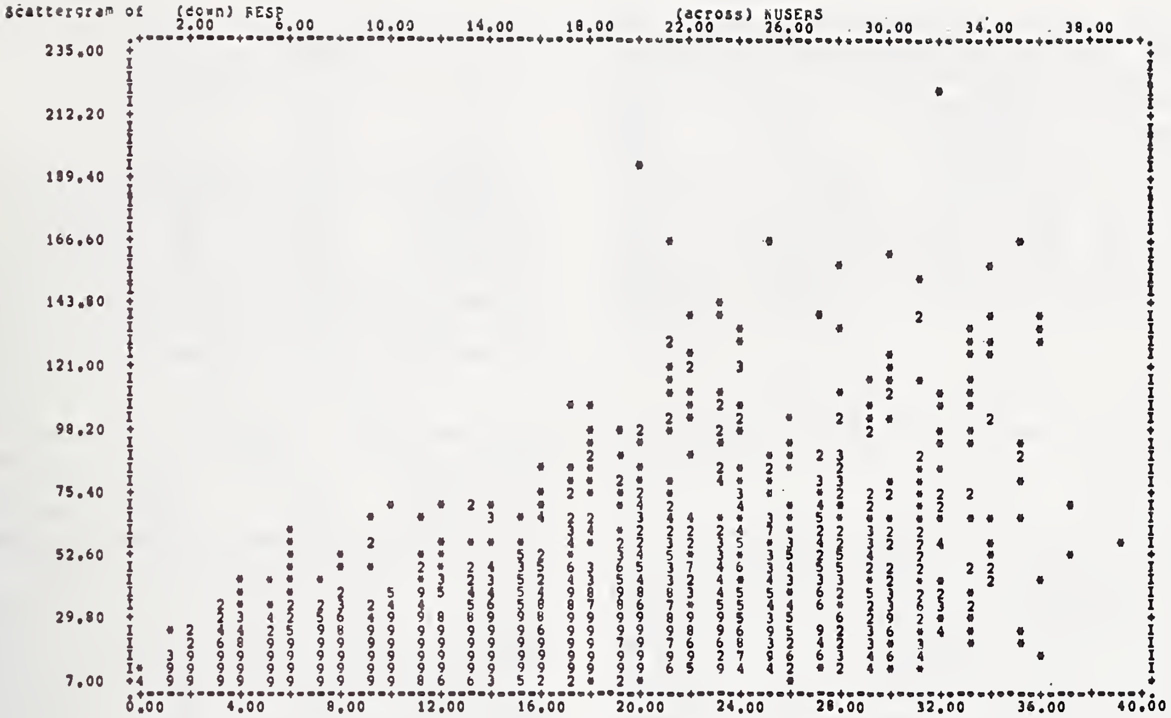


Figure 2: System VI. Scattergram of the task man man. Nu (NUSERS) versus rt (RESP).

In Figures 1, 2 and 3, we display the scattergram of one task measured in system VI, the text formating one called MAN MAN. The over all shape observed here was typical of each scattergram we made. nau (AUSERS) always tended to produce more compact distributions and np (PROCEC) had a larger range.

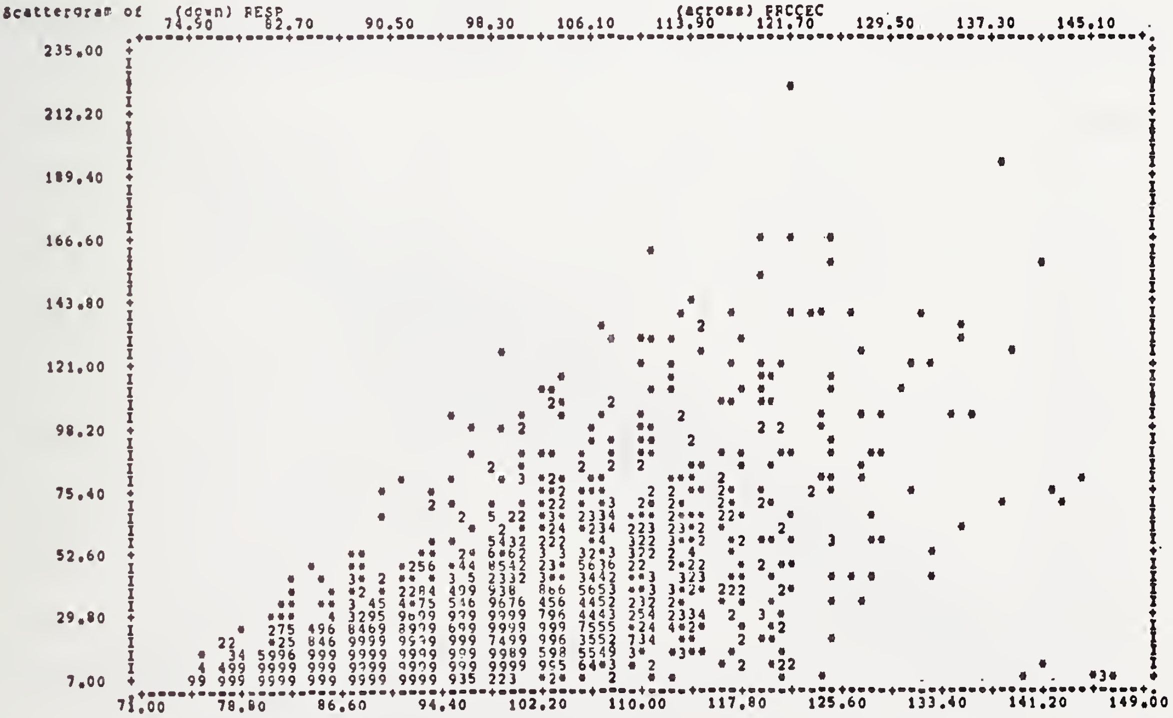


Figure 3: System VI. Scattergram of the task man man. Np (PROCEC) versus rt (RESP).

Two very important observations should be made from these figures. First, the variance of the response time distributions increases as the work load estimator increases. Second, the distributions show a large degree of skewness towards the larger values of the work load estimators. This means that large upper "tails" were observed. These two statistical characteristics of the behavior of response time applied consistently to all systems and tasks in script.

Figure 4 has the scattergram of nu (NUSERS) versus nau (AUSERS) observed in system P7A. The most interesting statistical feature of it is the existence of two clouds. The principal one, (i.e., the one in diagonal), was typical of all of these scattergrams. It graphically shows why the two estimators have such a high correlation coefficient. The secondary cloud, that parallel to the horizontal axis, is a small cluster of points which depicts the system with a large number of users logged in but very few doing work. It must be said that this did not occur often, because there are only 30 such measurements out of the 3206 we had for P7A. Most systems did not exhibit such a distinguishable secondary cloud.

3.1 Robustness Of Our Correlation Coefficients

The correlation between work load estimators is a function of the working habits of the installation's user community. Table 3 presents the correlation coefficients obtained from random subsamples in all systems. The size of the subsample and its percentage of the whole sample are indicated.

By comparing Table 3 with Table 2 we see that there is indeed great stability in these numbers. Another remarkable fact is that all relative orderings between coefficients are preserved. In each system the ordering of the correlations coincide. This clearly indicates a high degree of continuity in user habits. On the other hand, some hardware changes do bring different bahavior patterns as can be seen with P4B and P4A, VAB and VAA. From an overall analysis of the behavior of each installation which underwent hardware changes, we were able to detect changes in the user habits only when the hardware additions significantly altered the responsiveness of the system at all levels of the work load.

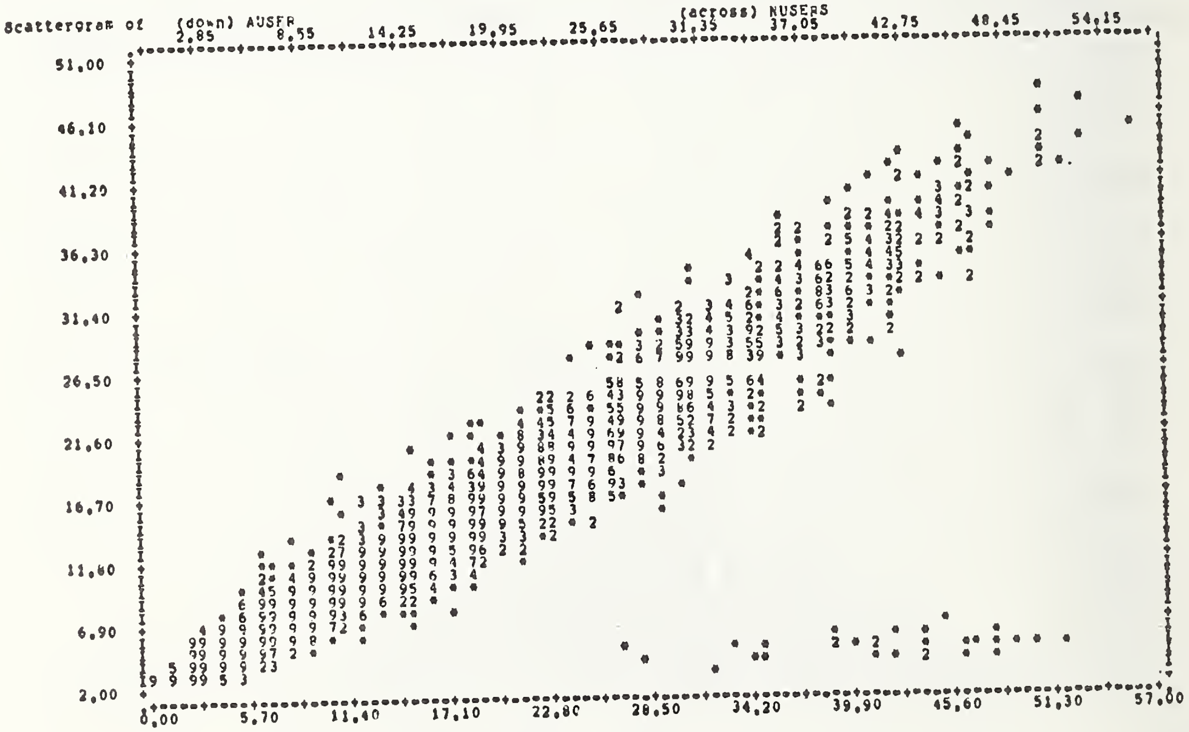


Figure 4: System P7A. Scattergram of nu (NUSERS) versus nau (AUSER).

	No. of	% total	Correlation Coefficients		
System	meas.	meas.	nu vs nau	nau vs np	nu vs np
VI	1708	49.78	.896	.785	.668
P7A	1471	49.71	.909	.849	.795
VM	740	50.04	.880	.860	.789
P7B	515	50.06	.948	.899	.885
VAA	425	50.03	.842	.821	.740
VC	398	50.31	.887	.776	.684
VB	225	51.72	.836	.861	.769
P4A	156	51.65	.705	.834	.698
P4B	114	48.30	.384	.343	.489
VAB	39	42.85	.339	.856	.481

Table 3: Correlation coefficients of subsamples.

3.2 Data Of Two Systems Mixed Up

As a case in point for always doing visual analysis of the data we present Figures 5, 6 and 7. A small hardware change, the addition of 8 ports to an installation, had gone unnoticed. We had labeled this system VA. Even though this change may seem unimportant, Table 4 shows the effect it had in the correlation coefficients. The regression analysis was affected accordingly. The two resulting systems were labeled VAB and VAA.

When analyzing Figure 5 we noticed and traced the hardware change. In the data for VAB and VAA we saw that most correlations improved. Moreover, the pair nau and np showed such a large increase that we chose it for display. Figure 6 has the scattergram for VAB and Figure 7 that for VAA. The new correlation values obtained for the two systems were in greater harmony with the other VAX measurements. The low value obtained for nu versus nau in VAB may be attributed to the use of software that began running in that system at the time [6,8].

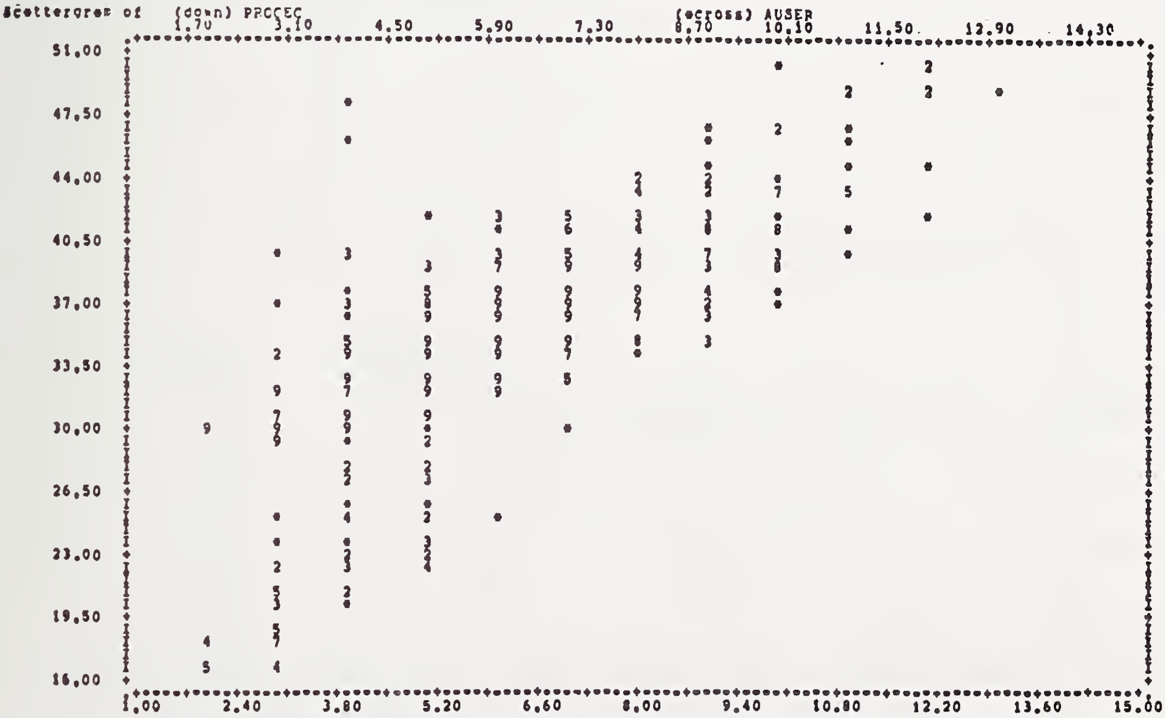


Figure 5: System VA. Scattergram of nau (AUSER) versus np (PROCEC).

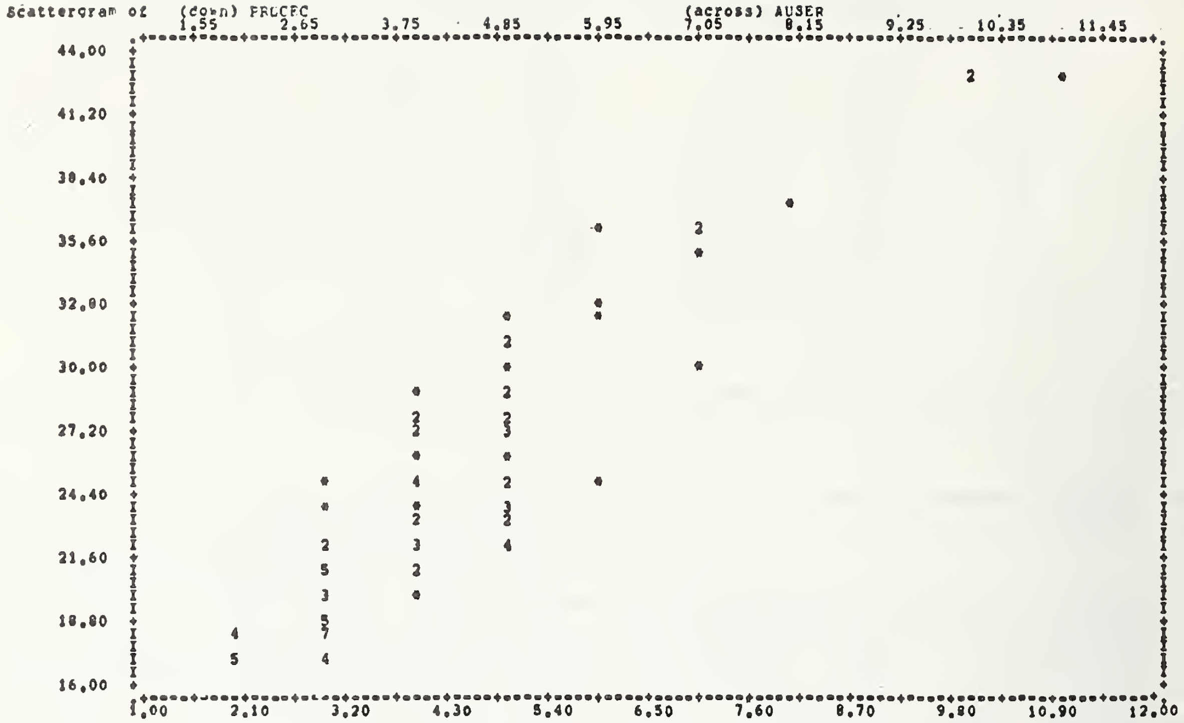


Figure 6: System VAB. Scattergram of nau (AUSER) versus np (PROCEC).

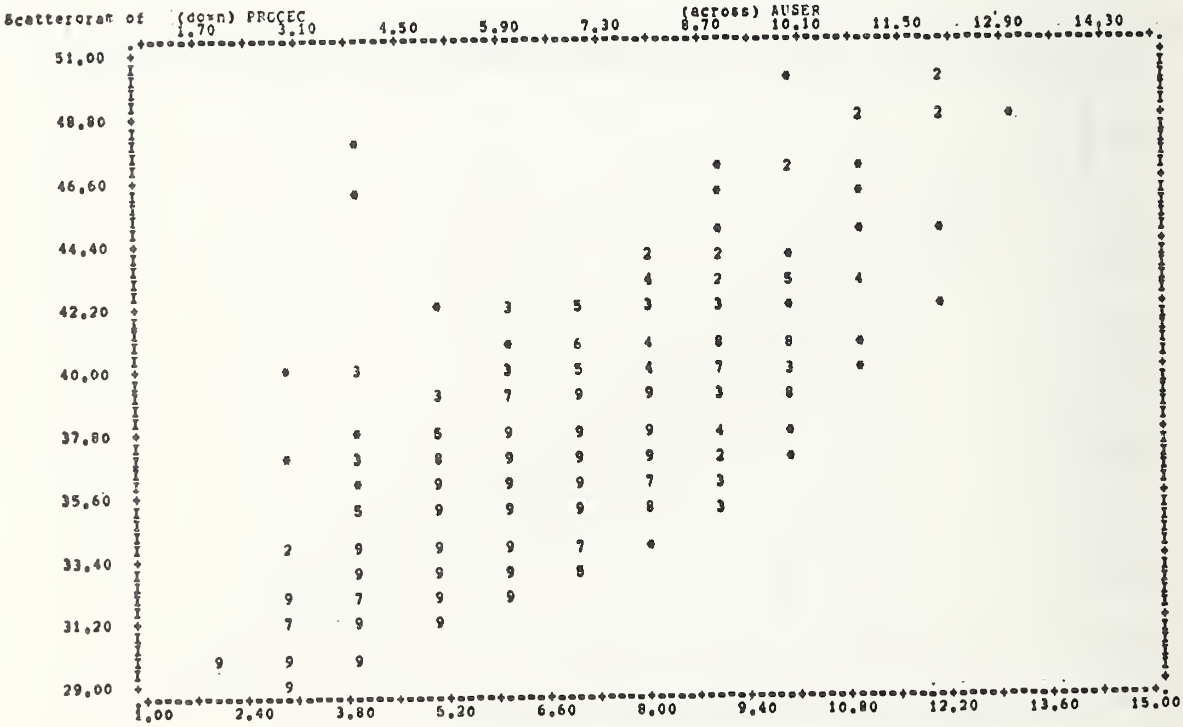


Figure 7: System VAA. Scattergram of nau (AUSER) versus np (PROCEC).

System	Correlation Coefficients nu vs nau	 nau vs np	 nu vs np
VA	.758	.769	.443
VAB	.492	.906	.606
VAA	.852	.838	.746

Table 4: Correlation Coefficients for mixed up systems.

4.0 ANALYSIS OF THE RESPONSE TIME DISTRIBUTIONS

In Section 3 we mentioned that the scattergrams of the different tasks in the different systems consistently showed that the distributions for response time had larger variances and larger degrees of skewness for higher values of the work load estimator.

Figures 8 and 9 are histograms, obtained using minitab [16], of the cpu-bound task in VI and the P7A systems respectively. They correspond to the measurements made when each of these systems had 10 users logged in. When done on a larger scale, with consequently less clustering, they show even larger tails. Inspection of these histograms clearly shows that response time does not follow a normal distribution.

```
EACH * REPRESENTS   2 OBSERVATIONS

MIDDLE OF    NUMBER OF
INTERVAL     OBSERVATIONS
     4.         0
     8.        53      ***************************
    12.        28      **************
    16.         8      ****
    20.         8      ****
    24.         4      **
    28.         6      ***
    32.         4      **
    36.         4      **
    40.         1      *
```

Figure 8: System VI. Clustered histogram of response time. Task cpu-bound. 10 users logged in.

```
MIDDLE OF    NUMBER OF
INTERVAL     OBSERVATIONS
     5.         2      **
    10.        44      ********************************************
    15.        19      *******************
    20.        10      **********
    25.         6      ******
    30.        12      ************
    35.         5      *****
    40.         5      *****
    45.         1      *
    50.         2      **
    55.         2      **
    60.         1      *
```

Figure 9: System P7A. Clustered histogram of response time. Task cpu-bound. 10 users logged in.

The ideal work load estimator is that which exactly determines the activities in a system. Response time measured against an ideal estimator will behave as a function; i.e., the variance of the measurements is zero or, in other words we always obtain the same value. It is clear that our estimators are fairly rough. Their main advantages and interest arise from their easy definition, their portability and the low overhead involved in their measurement. The problem then is to find suitable statistical distributions to fit our data from which we may obtain estimations of our desired performance parameters.

A thorough analysis of our measurements indicated that response time may always be accurately approximated by a Gamma distribution. This distribution offers the appropriate flexibility to adequately fit our data. Our problems are then reduced to characterize the distribution's parameters. The problem is that we need to fit one distribution for each value of the work load estimator. We also know that the variances are different. In full generality this requires finding too many parameters.

We then looked for possible relationships between the first and second moments of the observed distributions. Much to our surprise we found that there was a high correlation between the mean and the variance. Figures 10 and 11 depict the VAriance plotted against the AVerage for the systems VI and P7A (N7). The measurements were taken from the cpu-bound task using nau (AC) as work load estimator. The associated regression analysis showed that in VI 92% of the variation in the variance could be explained in terms of the variation in the mean. This was 68% for P7A. Other tasks and systems had the same behavior and their values were always above 60%. This empirical relationship provided the final complexity reduction needed to appropriately fit a Generalised Linear Model [12].

5.0 MODELLING DATA WITH GLIM IN FIVE SYSTEMS

GLIM is an interactive package for modelling data through generalised linear models [2,12]. Its utilization allows fitting different parameter combinations to the data in brief time. The theory behind the models is presented in Section 5.1. From the user's point of view, one defines a dependent variable, yvar, which for us corresponds to response time, and fits its expected value in terms of other observed variables. Whenever more than one such observed variable exists, as in our case where we have nu, np and nau,

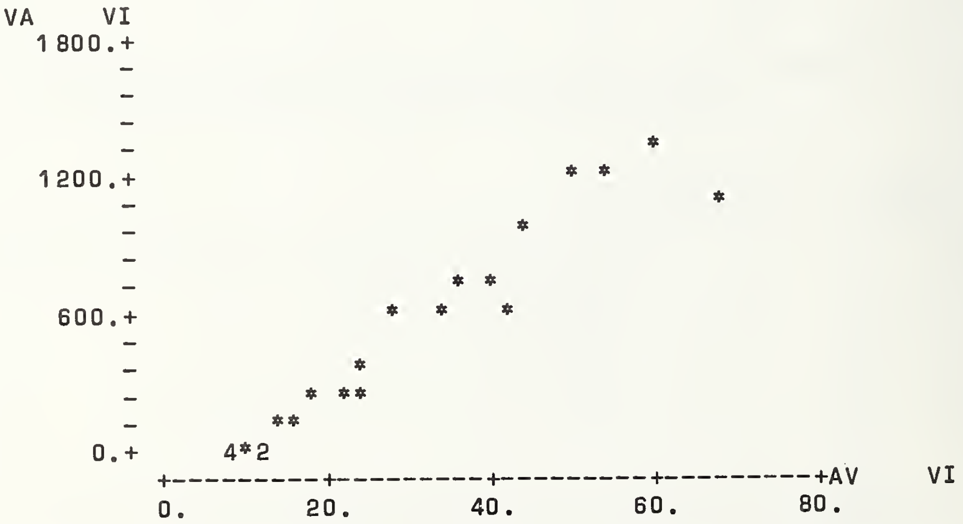


Figure 10: System VI. Mean versus variance. Task cpu-bound.
Work load estimator: nau.

```
  VA    N7
    18000.+
          -
          -
          -
          -                                 *
    12000.+                       *
          -
          -
          -                            *
          -
     6000.+                         *      *
          -
          -             **  ** *  *
          -           * *  **     *
          -    3   6 **              *
        0.+   343*
            +---------+---------+---------+---------+AV   N7
            0.       60.      120.      180.      240.
```

Figure 11: System P7A. Mean versus variance. Task cpu-bound. Work load estimator: nau.

linear combinations of them can also be fitted as in ordinary regression analysis.

Moreover, GLIM also permits choosing the underlying error distributions, and has different modelling assumptions for each such class. For example, normal errors are assumed to have the same variance. This is not the case for Gamma errors, where the variance is assumed to be a fixed multiple of the mean. The analysis presented in Section 4 supports this last hypothesis. Since we had observed that the distibutions had gamma shapes, and that the variances were directly proportional to the means, we assumed that GLIM errors were gamma.

The other degree of choice which GLIM offers is the relationship, or link, existing between the expected value of yvar and the estimating variables. Possibilities include the identity relation, the inverse, square root, and logarithm among others. If, for example, one chooses the logarithm relationship, then it means that the values of the fitting model and yvar are linked through the logarithm; i.e., the expected value of yvar is the exponentiation of the value given by the fitting model.

It is well known that response time as a function of work load estimators behaves exponentially [4,5]. Thus the logarithm link was used. As a criterion for evaluating alternative fitting models, we not only considered the deviance given by GLIM, which is the maximized likelihood, but also looked at the sums of squares of the differences between our estimation and the observations. GLIM minimizes this in the presence of normally distributed errors.

5.1 The Generalized Linear Models

Underlying the concept of a statistical model for a random variable is the idea that the variable under investigation has a definite structure which will explain the values actually obtained as well as predict future values. The structure is in fact a description of the population and will be mirrored in whatever sample we obtain. It postulates that the variable can be expressed in terms of other more basic variables: the components of the structure. If these latter variables have fixed (though possibly unknown) values they are termed systematic components whereas if they too are random variables they are termed random components.

Definition of a Generalized Linear Model

Even though one may consider any random variable Y, where y[i] denotes its ith sample value, to be representable by any combination of any number of components, for most practical purposes simple structures suffice. A Generalised Linear Model (GLM) is determined as follows. Let a set of independent random variables Y[i] (i = 1, ... , n) have means u[i], so that

Y[i] = u[i] + e[i] .

Then there are three basic properties which define a GLM:

The Error Structure

The probability density function of Y[i] is given by p(Y[i])=exp{(Y[i]Q[i] - b(Q[i]))/a[i](F) + c(Y[i],F)} for suitable choice of a[i], b and c. (Note that F, termed the scale parameter, is constant for all i.) The mean and the variance of Y[i] can then be expressed in terms of Q[i] and F:

E(Y[i]) = b'(Q[i])
var(Y[i]) = b''(Q[i]) * a[i](F)

where primes denote differentiation with respect to Q.

For example the Normal distribution is obtained by setting a[i](Q) = Q, b(Q[i]) = 1/2 * Q[i]**2 and c(Y[i],Q) = -1/2 * {log(2TQ) + Y[i]**2/F}, where F would usually be denoted by sigma squared.

It is convenient to write b''(Q[i]) = t[i]**2, the variance function, which is a function of u[i] only, so var(Y[i]) = a[i](Q) * t[i]**2 = Q * t[i]**2 / w[i] where the w[i] are called the prior weights, and the functions a[i](Q) have the form F/w[i].

The Linear Predictor

The role played by the remaining variables in the structure of each observation is expressed as a linear sum of their effects for the observation, called the linear predictor, n[i],

n[i] = sum{j=1 to p}x[i,j] * b[j]

where the x[i,j] are known and the b[j] are (usually unknown) parameters. The matrix X, of order n x p, is called the design matrix. The righthand side of the equation is called the linear structure. If an x[i,j] represents the presence or absence of a level of a factor then b[j] is the effect of that level; if x[i,j] is the value of a quantitative covariable then b[j] scales x[i,j] to give its effect on n[i].

The Link Function

The relationship between the mean of the ith observation and its linear predictor is given by the link function g[i]:

n[i] = g[i](u[i])

where the g[i] are assumed monotonic and differentiable. We define h[i] where

u[i] = h[i](n[i])

as the inverse of the link function. Although each observation could in theory have a different link function, this is rare in practice and so the subscript is dropped.

In summary, a particular GLM can be identified by specifying the error distribution of the random component, the make-up of the linear predictor and the function linking the means to the linear predictors. All error distributions must belong to the exponential family.

5.2 Summary Of The Analysis Of Five Systems

In this section we present the results obtained using GLIM on the data of five systems. We have chosen the modelling of response time for the C compilation task (cc) for display, because of the relevance that the C programming language has in all UNIX systems [11,15]. We display seven linear models based in the three work load estimators np, nu and nau.

Table 5 summarizes the deviances and sums of squares of the differences between the fitted points and the observed ones. The values given by GLIM have been divided by the degrees of freedom of the corresponding samples to take into account the size of the sample. We have named these quotients the normalized deviance and the normalized sum of squares.

With the exception of P7A, we see in Table 5 that in all systems the best single estimator was np. In P7A it turned out to be nau. We can also see that the best estimator pair was np+nu. Again P7A was anomalous in this respect. The scattergrams for rt versus each of our three work load estimators for P7A show that there is a large degree of variability. This system did not have the kind of behavior observed in the others which was akin to Figures 1, 2 and 3. Nau does group the measurements much better in this case. The model is correct when it gives nau as a better fit than np.

System		Linear Estimators						
		np	nu	nau	np+nu	np+nau	nu+nau	np+nu+nau
VI	Normalized: deviance	.1641	.1719	.1914	.1483	.1587	.1783	.1484
	Normalized: sum of sqr	24.47	25.66	26.79	22.22	23.26	26.68	22.22
P7A	Normalized: deviance	.4654	.3237	.3611	.3167	.333	.3135	.3085
	Normalized: sum of sqr	910.9	360.3	347.5	357.5	347.5	347.7	341.53
VM	Normalized: deviance	.175	.1875	.1977	.1609	.1732	.1854	.15636
	Normalized: sum of sqr	179.8	207.0	209.5	179.3	181.0	203.9	178.8
P7B	Normalized: deviance	.2408	.2566	.2887	.2334	.2410	.2546	.2157
	Normalized: sum of sqr	491.6	540.9	574.9	490.6	490.9	537.1	447.79
P4A	Normalized: deviance	.1132	.1463	.1656	.1091	.1125	.1402	.1069
	Normalized: sum of sqr	210.2	237.2	245.7	210.1	209.3	236.8	208.0

Table 5: Normalized deviance and normalized sums of squares for the different systems and linear estimators. Task: C compilation.

The failure of np to be the best single estimator indicates that the users in this system must have a peculiarity not present in other systems. This may be due to the fact that it is heavily used for instructional purposes. Since many users may be concurrently executing the same code, such as a toy operating system for example, severe distortions from the np estimator point of view may exist. For example, the memory utilization of several users executing the same shared piece of code is much less than it would be if each had his own copy. Thus the load on the resources of the machine is less than that suggested by the number of processes. This load is clearly more faithful to the number of active users.

Tables 6.1 and 6.2 have all the parameters given by GLIM in the analysis of the task cc. In Section 5.1 we saw how to obtain analytic formulae to represent the expected value of rt in terms of these parameters.

It is interesting to notice from Tables 6.1 and 6.2 that in several systems the addition of nau to the estimator np+nu does not increase the accuracy of the model. Moreover, there are other systems where the normalized deviance and the normalized sum of squares do improve but the coefficient associated when nau has a standard error which renders it not significant. These observations lead us to conclude that for most systems nau does not add significant modelling information in the presence of np and nu. In the case of a long data gathering period, after an exploratory period one could assess whether nau is indeed needed and if not omit gathering statistics about it. This will reduce the overhead of the method and its total cost, while preserving its modelling accuracy.

	SYSTEMS					
Parameters	VI		P7A		VM	
	Estimate	Std.Error	Estimate	Std.Error	Estimate	Std.Error
%GM	-0.8103	.5845E-1	-1.010	.7959E-1	.5254	.1008
np	.3029E-1	.6104E-3	.3135E-1	.6388E-3	.5679E-1	.2155E-2
scale	.1642		.4656		.1750	
%GM	1.533	.1329E-1	1.736	.1980E-1	2.748	.1933E-1
nu	.3962E-1	.8237E-3	.5547E-1	.8986E-3	.8015E-1	.2968E-2
scale	.1719		.3237		.1875	
%GM	.1721	.1150E-1	1.747	.2234E-1	2.467	.3164E-1
nau	.5028	.1213E-1	.6462E-1	.1200E-2	.1047	.4341E-2
scale	.1914		.3611		.1977	
%GM	-.314E-2	.6737E-1	1.143	.7197E-1	1.104	.1282
np	.1865E-2	.8011E-3	.5721E-2	.6688E-3	.3961E-1	.3075E-2
nu	.2136E-1	.1057E-2	.4937E-1	.1128E-2	.4032E-1	.4090E-2
scale	.1484		.3166		.1609	
%GM	-0.2587	.7452E-1	.6764	.7219E-1	.8455	.1446
np	.2310E-1	.8622E-3	.1001E-1	.6731E-3	.4586E-1	.3966E-2
nau	.1800	.1587E-1	.5380E-1	.1435E-2	.2758E-1	.7516E-2
scale	.1587		.3330		.1732	
%GM	1.544	.1319E-1	1.666	.2082E-1	2.628	.3608E-1
nu	.2973E-1	.1349E-2	.3582E-1	.2208E-2	.5818E-1	.6053E-2
nau	.1696	.1882E-1	.2672E-1	.2791E-2	.3467E-1	.8623E-2
scale	.1679		.3135		.1855	
%GM	.1097E-1	.7326E-1	1.174	.7104E-1	.7469	.1389
np	.1849E-1	.8680E-3	.4811E-2	.6636E-3	.5351E-1	.3770E-2
nu	.2098E-1	.1320E-2	.3161E-1	.2243E-2	.6612E-1	.5560E-2
nau	.9277E-2	.1917E-1	.2527E-1	.2783E-2	-.631E-1	.9850E-2
scale	.1484		.3085		.1564	

Table 6.1: GLM parameters for three systems and seven linear estimators. Task: C compilation.

	SYSTEMS			
Parameters	P 7 B		P 4 A	
	Estimate	Std.Error	Estimate	Std.Error
%GM	-1.229	.1529	.6347	.1491
np	.3951E-1	.1401E-2	.7122E-1	.4629E-2
scale	.2408		.1133	
%GM	2.060	.4108E-1	2.465	.4433E-1
nu	.4390E-1	.1626E-2	.1509	.1258E-1
scale	.2566		.1463	
%GM	2.030	.4724E-1	2.162	.8466E-1
nau	.5319E-1	.2201E-2	.1395	.1466E-1
scale	.2888		.1657	
%GM	-.1511	.2514	.9139	.1685
np	.2589E-1	.2892E-2	.5752E-1	.6052E-2
nu	.1728E-1	.3250E-2	.5109E-1	.1448E-1
scale	.2334		.1091	
%GM	-1.381	.2799	.5103	.1673
np	.4150E-1	.3342E-2	.8106E-1	.7384E-2
nau	-.321E-2	.4794E-2	-.344E-1	.1934E-1
scale	.2410		.1125	
%GM	2.112	.4508E-1	2.212	.7903E-1
nu	.5862E-1	.5897E-2	.1140	.1604E-1
nau	-.198E-1	.7526E-2	.6519E-1	.1757E-1
scale	.2546		.1402	
%GM	-0.9839	.2656	.7749	.1755
np	.3762E-1	.3166E-2	.7059E-1	.7633E-2
nu	.5266E-1	.5435E-2	.6126E-1	.1486E-1
nau	-.637E-1	.7889E-2	-.560E-1	.1955E-1
scale	.2157		.1068	

Table 6.2: GLM parameters for two systems and seven linear estimators. Task: C compilation.

5.3 Robustness Of The Results In Three Systems

In Tables 7, 8 and 9 we present the parameters given by GLIM when ramdom subsamples of different sizes of the same distribution were analyzed. We considered the same task as in Section 5.2: the C compilation. These subsamples were made from the total sample by a program which selected random entries with a fixed probability for membership. We created three files with membership probabilities of .5, .25 and .125, to check the robustness of the modelling technique.

From the appropriate entries in Tables 6.1 and 6.2 we may see that the coefficients found in Tables 7, 8 and 9 are very robust. In almost all of the cases (the exception being the 170-point subsample in System VM) the estimators for the smaller samples were within their standard error from those found in the larger samples.

	Sample Size 1497		Sample Size 729		Sample Size 376	
Parameter	Estimator	Std.Error	Estimator	Std.Error	Estimator	Std.Error
%GM	.5640	.1021	.6707	.1404	.6130	.1906
np	.112E-1	.9473E-3	.9783E-2	.1304E-2	.1069E-1	.1755E-2
nau	.5146E-1	.202E-2	.5608E-1	.2785E-2	.5173E-1	.4124E-2
scale	.3393		.3146		.3395	

Table 7: System P7B. Robustness of the selected linear estimator. Task: C compilation.

	Sample Size 1722		Sample Size 851		Sample Size 428	
Parameter	Estimator	Std.Error	Estimator	Std.Error	Estimator	Std.Error
%GM	-.1275	.9481E-2	-.6774E-2	.1405	.6659E-1	.1886
np	.2020E-1	.1128E-2	.1893E-1	.1652E-2	.1765E-1	.2264E-2
nu	.2008E-1	.1502E-2	.2032E-1	.2060E-2	.2341E-1	.2955E-2
scale	.1506		.1422		.1309	

Table 8: System VI. Robustness of the selected linear estimator. Task: C compilation.

	Sample Size 739		Sample Size 365		Sample Size 170	
Parameter	Estimator	Std.Error	Estimator	Std.Error	Estimator	Std.Error
%GM	1.348	.1828	1.428	.2424	-.5394E-1	.3657
np	.3401E-1	.4387E-1	.3209E-1	.5783E-2	.6752E-1	.8716E-2
nu	.4529E-1	.5835E-2	.4458E-1	.7294E-2	-.1081E-2	.1062E-1
scale	.1742		.1553		.1186	

Table 9: System VM. Robustness of the selected linear estimator. Task: C compilation.

However, the price paid for fewer data points was much larger standard errors of the fitting estimators. Given that each smaller sample was roughly half of the larger one, we see that standard errors diminished approximately 25% when the size of the sample doubled. This rule can be a guide in deciding the duration of the data gathering period as a function of the desired accuracy.

We also appreciated that for the best linear estimators in each system, those found from samples which had more than 300 points showed high levels of significance, i.e., the standard error of the estimators were small. This gives us an empirical global bound on the minimum number of data points one should gather in a system.

6.0 CONCLUSIONS

We have found that response time has a statistical behavior which is consistent enough to permit the utilization of generalized linear models (GLMs) to describe and predict its values. This, in turn, has validated the utilization of rather imprecise work load estimators, such as response time (rp), number of users (nu) and number of active users (nau), as basis for estimating selected performance indices in a computer installation.

Moreover, the robustness of the models also permit comparisons of different computer installations using the terminal probe method as data gathering technique while the systems are executing their natural work loads. We have also seen that a minimum of 300 points per installation appears to be necessary for obtaining a minimum of accuracy, and that doubling the size of the sample will reduce by 25% the standard error of the parameters estimated by the model. We may then assess the cost of achieving a predetermined error bound much better.

When measured against our work load estimators np, nu and nau, response time appears to have Gamma distributed values. What is more, we have empirically observed that the means and variances of these distributions are highly correlated as functions of each of our work load estimators. Their ratio is almost directly proportional to the values of the work load estimator. This behavior is essential when using a GLM with Gamma error, and gives us a better understanding of the behavior of response time.

We could also observe that for most systems and tasks, nau adds no significant additional information to that given by np and nu. This fact can be used to reduce the data gathering cost and the overhead of the terminal probe method.

ACKNOWLEDGEMENTS

I am very grateful to my colleage Dr. German Rodriguez who introduced me to GLIM, Minitab and several other tools and techniques. Without his patience and knowledge this work would have been considerably less thorough.

7.0 REFERENCES

1. Babaoglu, O., W. Joy and J. Porcar, ``Design and Implementation of the Berkeley Virtual Memory Extension to the UNIX Operating System,'' Department of EECS--Computer Science Division, University of California, Berkeley, (1979).

2. Baker, R. J., and Nelder, J. A., ``The GLIM System Manual, Release 3,'' Numerical Algorithms Group, NAG Central Office, Mayfield House, 256 Banbury Road, Oxford OX2 7DE, 1978.

3. Bourne, S. R., ``The UNIX Shell,'' The Bell System Technical J. 57, 6 Part 2 (Jul.-Aug. 1978), 1971-1990.

4. Cabrera, L. F., ``A Performance Analysis Study of UNIX,'' Proceedings of the Computer Performance Evaluation Users Group 16th Meeting, CPEUG 80, NBS Special Publication 500-65, Orlando, Florida, October 1980, pp. 233-243.

5. Cabrera, L. F., ``Benchmarking UNIX: A Comparative Study,'' in Experimental Computer Performance Evaluation (D. Ferrari and M. Spadoni eds.) North-Holland, Amsterdam, Netherlands,pp 205-215.

6. Cabrera, L. F., Paris, J.-F., ``Comparing User Response Times on Paged and Swapped UNIX by the Terminal Probe Method'', Proceedings of the Computer Performance Evaluation Users Group 17th Meeting, CPEUG 81, NBS Special Publication 500-83, San Antonio, Texas, November 1981, pp. 157-168.

7. Digital Equipment Corporation, VAX 11/780 Technical Summary. Maynard, Mass., 1978.

8. Fateman, R. J., ``Addendum to the Mathlab/MIT MACSYMA Reference Manual for VAX/UNIX VAXIMA,'' Department of EECS--Computer Science Division, University of California, Berkeley (Dec. 1979).

9. Ferrari, D. ``Computer Systems Performance Evaluation,'' Prentice-Hall, Englewood Cliffs, NJ, 1978.

10. Karush, A. D., ``The Benchmarking Method Applied to Time-Sharing Systems,'' Rept. SP-3347, System's Development Corporation, Santa Monica, CA, August 1969.

11. Kernighan, B. W. and J. R. Mashey, ``The UNIX Programming Environment,'' Computer 14, 4 (Apr. 1981), 12-24.

12. Nelder, J. A. and Wedderburn R.W.M., ``Generalized Linear Models,'' Journal of the Royal Statistical Society, A, 135, pp 370-384, 1972.

13. Nie, N. H. et al, ``SPSS, Statistical Package for the Social Sciences,'' Second Edition, McGraw-Hill, 1975.

14. Ritchie, D. M. and K. L. Thompson, ``The UNIX Time-Sharing System,'' Comm. ACM 17, 7 (Jul. 1974), 365-375. A revised version appeared in The Bell System Technical J. 57, 6 Part 2 (Jul.-Aug. 1978), 1295-1990.

15. Ritchie, D. M., S. C. Johnson, M.E. Lesk and B. W. Kernighan, ``The C Programming Language,'' The Bell System Technical J. 57, 6 Part 2 (Jul.-Aug. 1978), 1991-2019.

16. Ryan, T.A., Joiner, B. L. and Ryan B. F., ``MINITAB Reference Manual,'' Minitab Project, Statistics Department, 215 Pond Laboratory, Pennsylvania State University, University Park, PA 16802, November 1982.

SOME ELEMENTS OF SOFTWARE FUNCTION AND COST ANALYSIS AS RELATED TO PERFORMANCE

James E. Gaffney, Jr

Federal Systems Division
Gaithersburg, MD 20017

This paper provides some elements of modern software function and cost analysis. Proper emphasis on the software process is basic to ensuring that the software will perform as specified. Subjects covered are: the establishment of requirements, life cycle management and costing. They are all attributes of the software management process.

Keywords: Costing, life cycle management, requirements, and software management.

1. Requirements

Good communications between the software producer and the user is central to the realization of the software performance goals derived by the user. This means establishing what functions are to be provided and agreeing to cost, schedule, and quality objectives, and appropriate measures of them.

> "Good management of software involves both producer and user, and starts with a clear statement of the functions that the software is to provide and continues with a methodology that provides the technical controls and resource management to produce high quality software within acceptable funding and time limits."[1]

The software implementation process may be said to consist of four stages; the development of requirements, the creation of a system description, the creation of a design, and finally the writing of the code that will effect the portion of the requirements addressed by the software (others may be addressed by hardware per se). Each of these stages is increasingly specific in expressing desired functionality. The first three address people; only the last addresses the computer itself. The program implementation at each stage is a vehicle for communication of function. Each stage is an elaboration and implementation of the previous stage. Ideally, the progression from requirements to code is linear; actually, there is often feedback among the stages.

The establishment of requirements is often a very difficult task. Requirements include statements about the functionality to be provided, and also designation of the spatial and temporal performance objectives that the software is to satisfy. Proper attention to requirements is important because quality may be defined as "conformance to requirements."[2] Most generally, such "conformance" means that the product meets the needs of the user and satisfies stated performance criteria. Some specific measures of software quality that have been employed are:

° latent error content - defects present at time of delivery (estimated)

° mean time between failure

° number of defects found during some test period.

It can be said that "quality" is an aspect of "product integrity" which includes, among other aspects of the software development process, adherence to schedule and cost objectives.[3]

2. Life Cycle Management

The basic objective of software development management is to produce a desired amount of function at a specified quality level, within a given cost and schedule envelope. Appropriate quantitative and qualitative management techniques should be applied to identify and control the stages of the software development process and assure the quality of the resultant product. A quantitative management support system is suggested as an aid to the life cycle management process. Prior to the initiation of the development process, basic parameters of the software process and product including: amount of function (indicated by number of source lines of code or other appropriate measure), intended cost, and schedule should be estimated. During the development and maintenance/operations portions of the life cycle, parameters of the process and product can be monitored for adherence to objectives or requirements established earlier and corrections to the produce and product may be made as appropriate. At various stages of the life cycle, data about both the process and the product can be collected. This information can be used to make corrections to the estimation/prediction models, and more generally, to support the objective of learning from experience in a structured manner so that "it can be done better the next time." A key element of the software process is that there be a continuing interchange of data between the management support system and the (software) life cycle process.

3. Costing

The cost of software, both development and maintenance/operations, may be computed from the general relationship:

> labor (man months) = function to be developed x development rate

There are a number of measures of function 'size' available. Probably, the most commonly used is 'source lines of code' (SLOC). There are alternatives to this measure, such as 'function points' developed by Albrecht.[4] This measure relates the amount of "function" the software is to provide to the data it is to use (absorb) and to generate (produce). 'Function points' are relatable to SLOC.[5] Also, Britcher and Gaffney have suggested,[6] based on the state machine model of a software system,[7] another way of measuring the amount of function to be provided by a software system. They observe that any software system should have the same number of 'levels' of elaboration of function. Hence, one should be able to produce an estimate based on the number of "boxes" at a certain functional level in recognition that, on the average, about the same amount of function should be resident in a "box" at a given level in the specification hierarchy. The amount of function (say represented by SLOC) is a key parameter to be used by the performance analyst, even before the code has been written. Indeed, as Smith has described, "... a static analysis [can be used] to derive the mean, best case, and worst case response times. The static analysis is based on the optimistic assumption that there are no other jobs in the host configuration competing for resources."[8] This type of analysis deals with what has been termed monoprogramming performance.[9]

The 'development rate' specifies the rate (say in man months per thousand SLOC or per function point) at which the function is to be developed. The 'development rate' should be taken as the sum of the rates for each of the relevant activities or work components that constitute a particular software development process.[10] These components are the basis for a software engineering management model[11] used by the Federal Systems Division of IBM. Sixteen work components have been identified from which the software organization or the engineering organization involved in a software development project can structure its particular activities. They are:

° software requirements definition
° software system description
° software development planning
° engineering change analysis
° functional design
° program design
° test design
° software tools
° design evaluation
° module development
° development testing
° problem analysis and error correction
° software system test procedures
° software integration and test
° system test support
° acceptance test support

Thus, a cost estimate can be made by considering the nature of the particular software development job and the work components (such as program design, coding, etc.) that constitute it. Then, the labor (man months) for each component is estimated. The sum of these man month figures is the amount required for the given job. Considering the development process in terms of its constituents enables the estimator to achieve a greater degree of intellectual control that if he were to evaluate the process overall. For example, it may not be clear how the availability of a new process that facilitates unit testing would impact development productivity. However, its effect on the work component that covers unit test would be much easier to discern.[12]

The parameters of the model for estimating software development described above would be expected to be modified, based on the actual experience of the development organization for the type(s) of code concerned, as suggested in the section on life cycle management. This same model can support the management of maintenance and enhancement activities, treating them as a sequence of "development efforts." Depending upon the amount of change relative to the baseline system, then, the maintenance and operation's effort can be handled as a new project (albeit with a lot of retained code) rather than as a continuation of the old.[13]

4. Conclusions

Both the developers and users of software are concerned with issues of performance. Obtaining good software begins with the establishment of a proper set of requirements and continues with management that focuses on the development of the desired software functionality to be provided at a level of quality within an agreed upon cost and schedule envelope. There are tools and techniques available to support software management with quantitative assessments of the software process. They can contribute significantly to the software exhibiting the level of performance intended for it.

References

[1] NBS Special Publication 500-11, Computer Software Management, A Primer For Project Management and Quality Control, issued, July, 1977 (by Dennis W. Fife).

[2] Gaffney, J. E., Jr., Metrics In Software Quality Assurance, ACM '81, Los Angeles, November, 1981; pg. 3-31 (proceeding).

[3] Bersoff, E., Henderson, U., and Siegel, S. Software Configuration Management, An Investment In Product Integrity, Prentice-Hall, 1980.

[4] Albrecht, A. J., Measuring Application Development Productivity, Proceedings IBM Applications Development Symposium, Monterey, California; October, 1979; GUIDE International and SHARE, Inc., IBM Corporation, pg. 83.

[5] Albrecht, A.J., and Gaffney, J. E., Jr., Software Function, Source Lines of Code, and Development Effort Prediction - A Software Science Validation; to be published in the IEEE Transactions on Software Engineering; November, 1983.

[6] Britcher, R. M., and Gaffney, J. E., Estimates of Software Size From State Machine Designs; presented at Seventh Annual Software Engineering Workshop; NASA Goddard Space Flight Center; Greenbelt, MD; December, 1982.

[7] Ferrantino, A. B., and Mills, H. D., State Machines and Their Semantics in Software Engineering, IEEE COMSAC, Chicago, Fall, 1977.

[8] Smith, C. U. Increasing Information Systems Productivity Buy Software Performance Engineering, Proceeding of the 1981 Computer Measurement Group International Conference pg 5-14; New Orleans, December 1981.

[9] Dujmovic, J. J., Computer Selection and Criteria for Computer Performance Evaluation, International Journal of Computer and Information Sciences, Vol. 9; No. 6, 1980; pg. 435.

[10] Cruickshank, R. D., and Lesser, M., An Approach To Estimating and Controlling Software Development Costs in The Economics of Information Processing, R. Goldberg and H. Lorin (eds); Wiley, 1981.

[11] Quinnan, R. E., The Management of Software Engineering, Part V, IBM Systems Journal, Volume 19, No. 4, 1980.

[12] Gaffney, D. E., Jr. and Judge, R. W., The Quantitiative Impact of Four Factors on Work Rates Experienced During Software Development; Sixth Annual Software Engineering Workshop, NASA, Goddard Space Flight Center, Greenbelt, MD, Dec. 1981.

[13] Gaffney, J. E., Jr., A Macroanalysis Methodology for Assessment of Software Development Costs; in The Economics of Information Processing, Volume 2 (edited by R. Goldberg and H. Lorin), John Wiley and Sons, Inc., 1982.

BENCHMARK AND CONVERSION TOOL: TEST DATA REDUCTION PROGRAM

Frances A. Kazlauski

Naval Data Automation Command
Washington, D.C.

The Test Data Reduction Program (TDRP) is a software tool for use as a COBOL program conversion aid or benchmark testing aid. It ensures that newly developed, enhanced or converted COBOL programs are tested as thoroughly as required by employing existing production data files and utilizing fewer computer resources. It is used to extract appropriate data records from the production data file and create a reduced data file. The reduced data file will achieve the same level of testing coverage as the original production file and may be used for future program testing.

Key words: Benchmark; COBOL; conversion; coverage; extract; reduced.

1. Purpose

The purpose of this paper is to explain the capability and use of the Test Data Reduction Program (TDRP). TDRP is a software tool for the UNIVAC 1100 series and IBM compatible computers. It provides data reduction capability for sequential or indexed-sequential files.

2. Background

There are two common approaches to the preparation of test data: the generation of test data sets and the selection of a subset of records from an existing data file. The latter approach is more cost-effective for benchmarks or conversion projects. A method is needed which will extract a minimally required data set from an existing data file for thorough program testing. To assure that the extracted records perform the same coverage of testing as the original data file, program execution must be monitored through the insertion of probes. The implementation of such a capability provides a cost-effective vehicle for thoroughly testing benchmarks or converted software since testing can be done using smaller data files and less machine time. In addition, it also improves the reliability and validity of the tested software. The requirement for this capability led to the development of the Test Data Reduction Program (TDRP) in May 1980 at the Navy Regional Data Automation Center, Washington, D.C. The project sponsor is the Naval Data Automation Command (NAVDAC) Code 40.

3. Objectives

The basic objectives of TDRP are:

- o Extract records from an existing data file for program testing
- o Ensure the same coverage of program testing is achieved for the extracted data as for the original data
- o Achieve a "minimally-thorough-test" criterion

The "minimally-thorough-test" criterion ensures that each statement in a program be executed at least once. An additional feature of TDRP may be the development of more comprehensive test data for acceptance testing since the TDRP output report shows all untested code in the program.

4. Processing

TDRP is a test data extractor, not a test data generator. The program operates on a production system for which a data file already exists. It extracts records from this existing data file and creates a new test file based upon program logic and a user specified reduction factor. When a new logic path is taken, the current record is saved in the reduced file. Additional records may be saved to obtain the proper file size specified by the user.

TDRP sequential file processing consists of four phases: reformatting, instrumenting, executing and reporting. For indexed-sequential files which are randomly-accessed, an additional sort phase is required to eliminate duplicate records.

In the reformatting phase, the COBOL source program is put into the form required by the instrumentation phase. The reformatted program created in this phase contains one COBOL verb per line of code. A table of verb frequencies is also displayed at the end of the reformatted program. A portion of a reformatted COBOL source program is shown in Figure 1.

The instrumentation phase uses the reformatted COBOL source program and in addition reads the user parameter card file. The mandatory parameter card which specifies the data file to be reduced contains the following information:

- Maximum record length
- File name
- Record name
- Starting position for indexed key
- Key length for indexed files
- File type
- Blocking factor
- Percentage reduction factor (number between 0 and 50 where 0 indicates the minimally-thorough-test)

The user may provide optional parameter cards containing COBOL SELECT, FD and RECORD description clauses to describe the file to be reduced in the ENVIRONMENT and DATA divisions. If the SELECT card is present, the FD and RECORD clause cards must be coded. Once the parameter card file and reformatted source program are read in by the instrumentation phase, code is inserted in the reformatted program. This code enables the instrumented program to communicate with a data collection routine which collects execution statistics and monitors data record selection during execution. A portion of the instrumented code for the reformatted program in the previous figure in shown in Figure 2.

For randomly-accessed indexed sequential file processing, a skeleton COBOL program is read, modified and written to a file. This program will be executed later to sort the extracted data records.

In the execution phase, the instrumented COBOL program is compiled, mapped or linked, and executed. The TDRP subroutine for statistical data collection and data record extraction must be included in the collection process. During execution, the appropriate data records are written to the reduced file. In addition, execution statement counters are incremented and written to the statistical data collection file.

The reporting phase uses the reformatted COBOL source code and the statistical data file. The report lists the reformatted source code and the execution count for each statement. In addition, for conditional statements, the execution counts are also given for the "true" and "false" paths. This report may help in two ways:

- To evaluate the validity of the data file as a baseline to produce the reduced file
- To facilitate the creation of additional data records to assure completeness of program testing

A summary of program testing coverage is also included and consists of the following:

- Number of COBOL source statements
- Number of unexecuted statements
- Number of conditional statements
- Number of unexecuted conditional statements
- Percentage of unexecuted statements
- Percentage of unexecuted conditional statements

No provision is made to determine if the program logic is correct, however the report does show the statements executed utilizing the user data file. The correctness of the logic is difficult to check since recent studies show that most of the serious program errors are errors of omission due to incorrect or misinterpreted functional specifications or requirements (1)[1].

An example of a portion of a TDRP output report is illustrated in Figure 3.

5. Considerations

The potential savings in machine time and storage requirements can justify the cost of CPU time overhead, additional core requirements and the number of TDRP runs.

The CPU time overhead for compilation and execution is the result of the number of COBOL statements and subroutine calls inserted. The total number of inserted statements will be proportional to the number of paragraphs and conditional statements in the original program.

[1] Figures in parentheses indicate the literate references at the end of this paper.

```
CK-MSTR.
    IF IM-JOB EQUALS HOLD-JOB
    GO TO EVEN-MAT.
    IF IM-JOB IS GREATER THAN HOLD-JOB
    MOVE 1 TO CK-CD
    GO TO
    UNEVEN.
    MOVE IN-MSTR TO OT-MSTR.
    IF OM-3 EQUALS 5 AND OM-4 IS LESS THAN TODAYS-DT
    MOVE 3 TO
    OM-3
    PERFORM PR-PUN.
    IF OM-3 EQUALS 5 AND OM-5 IS LESS THAN TODAYS-DT
    MOVE 4 TO
    OM-3
    PERFORM PR-PUN.
    IF OM-6 EQUALS 8 AND OM-7 IS LESS THAN TODAYS-DT
    MOVE 7 TO
    OM-6
    PERFORM PR-PUN.
    IF OM-6 EQUALS 8 AND OM-8 IS LESS THAN TODAYS-DT
    MOVE 9 TO
    OM-6
    PERFORM PR-PUN.
    IF OM-JOB IS UNEQUAL TO SPACES
    WRITE OT-MSTR
    ADD 1 TO MSTR-1100-CT.
    GO TO CK-WHIP.
EVEN-MAT.
    MOVE IN-MSTR TO OT-MSTR.
    MOVE 3 TO CK-CD.
UNEVEN.
    IF H-ONE IS UNEQUAL TO SPACES
    PERFORM MOV-H1.
    IF H-TWO IS UNEQUAL TO SPACES
    PERFORM MOV-H2.
    IF H-THREE IS UNEQUAL TO SPACES
    PERFORM MOV-H3 THRU EX-1.
    IF H-FOUR IS UNEQUAL TO SPACES
    PERFORM MOV-H4 THRU EX-1.
    IF OM-JOB IS UNEQUAL TO SPACES
    PERFORM PR-PUN.
    IF OM-J1B EQUALS 3
    MOVE OM-J1C TO OM-36A
    MOVE OM-J1A TO
    OM-36B
    ELSE
    MOVE OM-JOB TO OM-36.
    MOVE SPACES TO H-ONE H-TWO H-THREE H-FOUR.
    IF OM-JOB IS UNEQUAL TO SPACES
    WRITE OT-MSTR
    ADD 1 TO MSTR-1100-CT.
CK-WHIP.
    IF OM-JOB EQUALS HOLD-W-JOB
    GO TO MAT-WHIP.
    IF HOLD-W-JOB IS LESS THAN OM-JOB
    GO TO LO-WHIP.
    MOVE SPACES TO OT-MSTR.
    IF CK-CD EQUALS 1
    GO TO MOV1.
    IF CK-CD EQUALS 3
    ALTER 1-SW
    TO PROCEED TO MOV1
    ELSE
    ALTER 1-SW
```

Figure 1. Reformatted COBOL Source Program

```
    TO PROCEED TO CK-MSTR.
    MOVE 0 TO CK-CD.
    GO TO READ-M.
MAT-WHIP.
    MOVE SPACES TO OT-MSTR.
    IF CK-CD EQUALS 1
    ALTER 2-SW
    TO PROCEED TO MOV1
    GO TO
    WHIP-MOV.
    IF CK-CD EQUALS 3
    ALTER 2-SW
    TO PROCEED TO MOV1
    ELSE
    ALTER 2-SW
    TO PROCEED TO CK-MSTR.
    ALTER 1-SW
    TO PROCEED TO WHIP-MOV.
    MOVE ZERO TO CK-CD.
    GO TO READ-M.
LO-WHIP.
    IF TBLE-CTR IS LESS THAN 999
    ADD 001 TO TBLE-CTR.
    MOVE HOLD-W-JOB TO TAB-JOB (TBLE-CTR).
LO-W-RTN.
    ALTER 2-SW
    TO PROCEED TO CK-WHIP.
    GO TO WHIP-MOV.
CLOS-MSTR.
    IF S-REC EQUALS HIGH-VALUES AND CR-IN EQUALS HIGH-VALUES
    GO
    TO FIN-CL.
    MOVE HIGH-VALUES TO IN-MSTR.
    GO TO 1-SW.
CLOS-WHIP.
    IF S-REC EQUALS HIGH-VALUES AND IN-MSTR EQUALS HIGH-VALUES
    GO TO FIN-CL.
    MOVE HIGH-VALUES TO CR-IN.
    GO TO 1A-SW.
CLOS-DET.
    IF IN-MSTR EQUALS HIGH-VALUES AND CR-IN EQUALS HIGH-VALUES
    GO TO FIN-CL-DET.
    MOVE HIGH-VALUES TO S-REC.
    GO TO 3-SW.
FIN-CL-DET.
    IF H-ONE EQUALS SPACES AND H-TWO EQUALS SPACES AND H-THREE
    EQUALS SPACES AND H-FOUR EQUALS SPACES
    GO TO FIN-CL.
    IF H-ONE EQUALS HIGH-VALUES
    GO TO FIN-CL.
    PERFORM UNEVEN.
FIN-CL.
    CLOSE MSTR-IN CR-FILE MSTR-OT PUNCH-OT
    DISPLAY '1100 MASTER RCDS ' MSTR-1100-CT UPON PRINTER.
    MOVE 99 TO LIN-CTR.
    MOVE ZEROES TO PG-CTR.
WR-WHIP.
    IF CTR-1 EQUALS TBLE-CTR
    GO TO LAST-CLOS.
    IF LIN-CTR IS GREATER THAN 50
    PERFORM WHIP-HD.
    ADD 001 TO CTR-1.
    MOVE TAB-JOB (CTR-1) TO PW-JOB1.
    IF CTR-1 EQUALS TBLE-CTR
```

Figure 1. Reformatted COBOL Source Program
(Continued)

```
     GO TO LAST-CLOS.
     ADD 001 TO CTR-1.
     MOVE TAB-JOB (CTR-1) TO PW-JOB2.
     IF CTR-1 EQUALS TBLE-CTR
     GO TO LAST-CLOS.
     ADD 001 TO CTR-1.
     MOVE TAB-JOB (CTR-1) TO PW-JOB3.
     IF CTR-1 EQUALS TBLE-CTR
     GO TO LAST-CLOS.
     ADD 001 TO CTR-1.
     MOVE TAB-JOB (CTR-1) TO PW-JOB4.
     IF CTR-1 EQUALS TBLE-CTR
     GO TO LAST-CLOS.
     ADD 001 TO CTR-1.
     MOVE TAB-JOB (CTR-1) TO PW-JOB5.
     IF CTR-1 EQUALS TBLE-CTR
     GO TO LAST-CLOS.
     ADD 001 TO CTR-1.
     MOVE TAB-JOB (CTR-1) TO PW-JOB6.
     IF CTR-1 EQUALS TBLE-CTR
     GO TO LAST-CLOS.
     ADD 001 TO CTR-1.
     MOVE TAB-JOB (CTR-1) TO PW-JOB7.
     IF CTR-1 EQUALS TBLE-CTR
     GO TO LAST-CLOS.
     ADD 001 TO CTR-1.
     MOVE TAB-JOB (CTR-1) TO PW-JOB8.
     IF CTR-1 EQUALS TBLE-CTR
     GO TO LAST-CLOS.
     ADD 001 TO CTR-1.
     MOVE TAB-JOB (CTR-1) TO PW-JOB9.
     WRITE P-REC BEFORE ADVANCING 2 LINES.
     MOVE SPACES TO P-REC.
     ADD 02 TO LIN-CTR.
     GO TO WR-WHIP.
 LAST-CLOS.
     WRITE P-REC BEFORE 01.
     CLOSE P-FILE.
     STOP RUN.
 WHIP-HD.
     MOVE SPACES TO P-REC.
     WRITE P-REC BEFORE ADVANCING NEXT-PG.
     ADD 0001 TO PG-CTR.
     MOVE 'LISTING OF JOBS ON LINK NO/JOB ORDER FILE BUT NOT ON
-    'MAST' TO P-HD2-W.
     MOVE 'ER JOB ORDER FILE FOR ' TO P-HD2A-W.
     MOVE TODAYS-DT TO P-DT-W.
     MOVE 'PAGE ' TO P-W-PG.
     MOVE PG-CTR TO P-W-ID.
     WRITE P-REC BEFORE ADVANCING 2 LINES.
     MOVE SPACES TO P-REC.
     MOVE ' JOB-NO   JOB-NO   JOB-NO   JOB-NO   J
-    'OB ' TO P-HD2.
     MOVE '-NO  JOB-NO   JOB-NO   JOB-NO   JOB-NO
-    ' ' TO P-HD2A.
     WRITE P-REC BEFORE ADVANCING 2 LINES.
     MOVE SPACES TO P-REC.
     MOVE 04 TO LIN-CTR.


 *** REFORMAT PROGRAM ENDED ***
```

Figure 1. Reformatted COBOL Source Program
(Continued)

```
TABLE OF VERB FREQUENCIES

VERB              COUNT


ADD                  19
ALTER                 9
CLOSE                 5
DISPLAY               2
ELSE                 17
END                   6
EXAMINE               2
EXIT                  2
GO                   78
IF                  105
INPUT                 4
MOVE                175
NEXT                 14
ON                    1
OPEN                  5
OUTPUT                3
PERFORM              14
READ                  5
RELEASE               2
RETURN                1
SORT                  1
STOP                  2
TALLYING              2
WRITE                17
```

Figure 1. Reformatted COBOL Source Program
(Continued)

```
CK-MSTR.
    MOVE   119 TO CURR-INDEX, CALL 'TDRP20'.
    MOVE   120 TO CURR-INDEX, CALL 'TDRP20'
    IF IM-JOB EQUALS HOLD-JOB
    MOVE   120 TO CURR-INDEX, CALL 'TDRP30'
    GO TO EVEN-MAT.
    MOVE   121 TO CURR-INDEX, CALL 'TDRP20'.
    IF IM-JOB IS GREATER THAN HOLD-JOB
    MOVE   121 TO CURR-INDEX, CALL 'TDRP30'
    MOVE 1 TO CK-CD
    GO TO
    UNEVEN.
    MOVE   122 TO CURR-INDEX, CALL 'TDRP20'.
    MOVE IN-MSTR TO OT-MSTR.
    MOVE   123 TO CURR-INDEX, CALL 'TDRP20'
    IF OM-3 EQUALS 5 AND OM-4 IS LESS THAN TODAYS-DT
    MOVE   123 TO CURR-INDEX, CALL 'TDRP30'
    MOVE 3 TO
    OM-3
    PERFORM PR-PUN.
    MOVE   124 TO CURR-INDEX, CALL 'TDRP20'.
    IF OM-3 EQUALS 5 AND OM-5 IS LESS THAN TODAYS-DT
    MOVE   124 TO CURR-INDEX, CALL 'TDRP30'
    MOVE 4 TO
    OM-3
    PERFORM PR-PUN.
    MOVE   125 TO CURR-INDEX, CALL 'TDRP20'.
    IF OM-6 EQUALS 8 AND OM-7 IS LESS THAN TODAYS-DT
    MOVE   125 TO CURR-INDEX, CALL 'TDRP30'
    MOVE 7 TO
    OM-6
    PERFORM PR-PUN.
    MOVE   126 TO CURR-INDEX, CALL 'TDRP20'.
    IF OM-6 EQUALS 8 AND OM-8 IS LESS THAN TODAYS-DT
    MOVE   126 TO CURR-INDEX, CALL 'TDRP30'
    MOVE 9 TO
    OM-6
    PERFORM PR-PUN.
    MOVE   127 TO CURR-INDEX, CALL 'TDRP20'.
    IF OM-JOB IS UNEQUAL TO SPACES
    MOVE   127 TO CURR-INDEX, CALL 'TDRP30'
    WRITE OT-MSTR
    ADD 1 TO MSTR-1100-CT.
    MOVE   128 TO CURR-INDEX, CALL 'TDRP20'.
    GO TO CK-WHIP.
EVEN-MAT.
    MOVE   129 TO CURR-INDEX, CALL 'TDRP20'.
    MOVE IN-MSTR TO OT-MSTR.
    MOVE 3 TO CK-CD.
UNEVEN.
    MOVE   130 TO CURR-INDEX, CALL 'TDRP20'.
    MOVE   131 TO CURR-INDEX, CALL 'TDRP20'
    IF H-ONE IS UNEQUAL TO SPACES
    MOVE   131 TO CURR-INDEX, CALL 'TDRP30'
    PERFORM MOV-H1.
    MOVE   132 TO CURR-INDEX, CALL 'TDRP20'.
    IF H-TWO IS UNEQUAL TO SPACES
    MOVE   132 TO CURR-INDEX, CALL 'TDRP30'
    PERFORM MOV-H2.
    MOVE   133 TO CURR-INDEX, CALL 'TDRP20'.
    IF H-THREE IS UNEQUAL TO SPACES
    MOVE   133 TO CURR-INDEX, CALL 'TDRP30'
    PERFORM MOV-H3 THRU EX-1.
    MOVE   134 TO CURR-INDEX, CALL 'TDRP20'.
    IF H-FOUR IS UNEQUAL TO SPACES
    MOVE   134 TO CURR-INDEX, CALL 'TDRP30'
```

Figure 2. Instrumented COBOL Source Program

```
    PERFORM MOV-H4 THRU EX-1.
    MOVE   135 TO CURR-INDEX, CALL 'TDRP20'.
    IF OM-JOB IS UNEQUAL TO SPACES
    MOVE   135 TO CURR-INDEX, CALL 'TDRP30'
    PERFORM PR-PUN.
    MOVE   136 TO CURR-INDEX, CALL 'TDRP20'.
    IF OM-J1B EQUALS 3
    MOVE   136 TO CURR-INDEX, CALL 'TDRP30'
    MOVE OM-J1C TO OM-36A
    MOVE OM-J1A TO
    OM-36B
    ELSE
    MOVE   137 TO CURR-INDEX, CALL 'TDRP20'
    MOVE OM-JOB TO OM-36.
    MOVE   138 TO CURR-INDEX, CALL 'TDRP20'.
    MOVE SPACES TO H-ONE H-TWO H-THREE H-FOUR.
    MOVE   139 TO CURR-INDEX, CALL 'TDRP20'
    IF OM-JOB IS UNEQUAL TO SPACES
    MOVE   139 TO CURR-INDEX, CALL 'TDRP30'
    WRITE OT-MSTR
    ADD 1 TO MSTR-1100-CT.
CK-WHIP.
    MOVE   140 TO CURR-INDEX, CALL 'TDRP20'.
    MOVE   141 TO CURR-INDEX, CALL 'TDRP20'
    IF OM-JOB EQUALS HOLD-W-JOB
    MOVE   141 TO CURR-INDEX, CALL 'TDRP30'
    GO TO MAT-WHIP.
    MOVE   142 TO CURR-INDEX, CALL 'TDRP20'.
    IF HOLD-W-JOB IS LESS THAN OM-JOB
    MOVE   142 TO CURR-INDEX, CALL 'TDRP30'
    GO TO LO-WHIP.
    MOVE   143 TO CURR-INDEX, CALL 'TDRP20'.
    MOVE SPACES TO OT-MSTR.
    MOVE   144 TO CURR-INDEX, CALL 'TDRP20'
    IF CK-CD EQUALS 1
    MOVE   144 TO CURR-INDEX, CALL 'TDRP30'
    GO TO MOV1.
    MOVE   145 TO CURR-INDEX, CALL 'TDRP20'.
    IF CK-CD EQUALS 3
    MOVE   145 TO CURR-INDEX, CALL 'TDRP30'
    ALTER 1-SWXXX
    TO PROCEED TO MOV1
    ELSE
    MOVE   146 TO CURR-INDEX, CALL 'TDRP20'
    ALTER 1-SWXXX
    TO PROCEED TO CK-MSTR.
    MOVE   147 TO CURR-INDEX, CALL 'TDRP20'.
    MOVE 0 TO CK-CD.
    GO TO READ-M.
MAT-WHIP.
    MOVE   148 TO CURR-INDEX, CALL 'TDRP20'.
    MOVE SPACES TO OT-MSTR.
    MOVE   149 TO CURR-INDEX, CALL 'TDRP20'
    IF CK-CD EQUALS 1
    MOVE   149 TO CURR-INDEX, CALL 'TDRP30'
    ALTER 2-SWXXX
    TO PROCEED TO MOV1
    GO TO
    WHIP-MOV.
    MOVE   150 TO CURR-INDEX, CALL 'TDRP20'.
    IF CK-CD EQUALS 3
    MOVE   150 TO CURR-INDEX, CALL 'TDRP30'
    ALTER 2-SWXXX
    TO PROCEED TO MOV1
```

Figure 2. Instrumented COBOL Source Program
(Continued)

```
    ELSE
    MOVE   151 TO CURR-INDEX, CALL 'TDRP20'
    ALTER 2-SWXXX
    TO PROCEED TO CK-MSTR.
    MOVE   152 TO CURR-INDEX, CALL 'TDRP20'.
    ALTER 1-SWXXX
    TO PROCEED TO WHIP-MOV.
    MOVE ZERO TO CK-CD.
    GO TO READ-M.
LO-WHIP.
    MOVE   153 TO CURR-INDEX, CALL 'TDRP20'.
    MOVE   154 TO CURR-INDEX, CALL 'TDRP20'
    IF TBLE-CTR IS LESS THAN 999
    MOVE   154 TO CURR-INDEX, CALL 'TDRP30'
    ADD 001 TO TBLE-CTR.
    MOVE   155 TO CURR-INDEX, CALL 'TDRP20'.
    MOVE HOLD--W-JOB TO TAB-JOB (TBLE-CTR).
LO-W-RTN.
    MOVE   156 TO CURR-INDEX, CALL 'TDRP20'.
    ALTER 2-SWXXX
    TO PROCEED TO CK-WHIP.
    GO TO WHIP-MOV.
CLOS-MSTR.
    MOVE   157 TO CURR-INDEX, CALL 'TDRP20'.
    MOVE   158 TO CURR-INDEX, CALL 'TDRP20'
    IF S-REC EQUALS HIGH-VALUES AND CR-IN EQUALS HIGH-VALUES
    MOVE   158 TO CURR-INDEX, CALL 'TDRP30'
    GO
    TO FIN-CL.
    MOVE   159 TO CURR-INDEX, CALL 'TDRP20'.
    MOVE HIGH-VALUES TO IN-MSTR.
    GO TO 1-SW.
CLOS-WHIP.
    MOVE   160 TO CURR-INDEX, CALL 'TDRP20'.
    MOVE   161 TO CURR-INDEX, CALL 'TDRP20'
    IF S-REC EQUALS HIGH-VALUES AND IN-MSTR EQUALS HIGH-VALUES
    MOVE   161 TO CURR-INDEX, CALL 'TDRP30'
    GO TO FIN-CL.
    MOVE   162 TO CURR-INDEX, CALL 'TDRP20'.
    MOVE HIGH-VALUES TO CR-IN.
    GO TO 1A-SW.
CLOS-DET.
    MOVE   163 TO CURR-INDEX, CALL 'TDRP20'.
    MOVE   164 TO CURR-INDEX, CALL 'TDRP20'
    IF IN-MSTR EQUALS HIGH-VALUES AND CR-IN EQUALS HIGH-VALUES
    MOVE   164 TO CURR-INDEX, CALL 'TDRP30'
    GO TO FIN-CL-DET.
    MOVE   165 TO CURR-INDEX, CALL 'TDRP20'.
    MOVE HIGH-VALUES TO S-REC.
    GO TO 3-SW.
FIN-CL-DET.
    MOVE   166 TO CURR-INDEX, CALL 'TDRP20'.
    MOVE   167 TO CURR-INDEX, CALL 'TDRP20'
    IF H-ONE EQUALS SPACES AND H-TWO EQUALS SPACES AND H-THREE
    EQUALS SPACES AND H-FOUR EQUALS SPACES
    MOVE   167 TO CURR-INDEX, CALL 'TDRP30'
    GO TO FIN-CL.
    MOVE   168 TO CURR-INDEX, CALL 'TDRP20'.
    IF H-ONE EQUALS HIGH-VALUES
    MOVE   168 TO CURR-INDEX, CALL 'TDRP30'
    GO TO FIN-CL.
    MOVE   169 TO CURR-INDEX, CALL 'TDRP20'.
    PERFORM UNEVEN.
```

Figure 2. Instrumented COBOL Source Program
(Continued)

```
FIN-CL.
    MOVE   170 TO CURR-INDEX, CALL 'TDRP20'.
    CLOSE MSTR-IN CR-FILE MSTR-OT PUNCH-OT.
    DISPLAY '1100 MASTER RCDS ' MSTR-1100-CT UPON PRINTER.
    MOVE 99 TO LIN-CTR.
    MOVE ZEROES TO PG-CTR.
WR-WHIP.
    MOVE   171 TO CURR-INDEX, CALL 'TDRP20'.
    MOVE   172 TO CURR-INDEX, CALL 'TDRP20'
    IF CTR-1 EQUALS TBLE-CTR
    MOVE   172 TO CURR-INDEX, CALL 'TDRP30'
    GO TO LAST-CLOS.
    MOVE   173 TO CURR-INDEX, CALL 'TDRP20'.
    IF LIN-CTR IS GREATER THAN 50
    MOVE   173 TO CURR-INDEX, CALL 'TDRP30'
    PERFORM WHIP-HD.
    MOVE   174 TO CURR-INDEX, CALL 'TDRP20'.
    ADD 001 TO CTR-1.
    MOVE TAB-JOB (CTR-1) TO PW-JOB1.
    MOVE   175 TO CURR-INDEX, CALL 'TDRP20'
    IF CTR-1 EQUALS TBLE-CTR
    MOVE   175 TO CURR-INDEX, CALL 'TDRP30'
    GO TO LAST-CLOS.
    MOVE   176 TO CURR-INDEX, CALL 'TDRP20'.
    ADD 001 TO CTR-1.
    MOVE TAB-JOB (CTR-1) TO PW-JOB2.
    MOVE   177 TO CURR-INDEX, CALL 'TDRP20'
    IF CTR-1 EQUALS TBLE-CTR
    MOVE   177 TO CURR-INDEX, CALL 'TDRP30'
    GO TO LAST-CLOS.
    MOVE   178 TO CURR-INDEX, CALL 'TDRP20'.
    ADD 001 TO CTR-1.
    MOVE TAB-JOB (CTR-1) TO PW-JOB3.
    MOVE   179 TO CURR-INDEX, CALL 'TDRP20'
    IF CTR-1 EQUALS TBLE-CTR
    MOVE   179 TO CURR-INDEX, CALL 'TDRP30'
    GO TO LAST-CLOS.
    MOVE   180 TO CURR-INDEX, CALL 'TDRP20'.
    ADD 001 TO CTR-1.
    MOVE TAB-JOB (CTR-1) TO PW-JOB4.
    MOVE   181 TO CURR-INDEX, CALL 'TDRP20'
    IF CTR-1 EQUALS TBLE-CTR
    MOVE   181 TO CURR-INDEX, CALL 'TDRP30'
    GO TO LAST-CLOS.
    MOVE   182 TO CURR-INDEX, CALL 'TDRP20'.
    ADD 001 TO CTR-1.
    MOVE TAB-JOB (CTR-1) TO PW-JOB5.
    MOVE   183 TO CURR-INDEX, CALL 'TDRP20'
    IF CTR-1 EQUALS TBLE-CTR
    MOVE   183 TO CURR-INDEX, CALL 'TDRP30'
    GO TO LAST-CLOS.
    MOVE   184 TO CURR-INDEX, CALL 'TDRP20'.
    ADD 001 TO CTR-1.
    MOVE TAB-JOB (CTR-1) TO PW-JOB6.
    MOVE   185 TO CURR-INDEX, CALL 'TDRP20'
    IF CTR-1 EQUALS TBLE-CTR
    MOVE   185 TO CURR-INDEX, CALL 'TDRP30'
    GO TO LAST-CLOS.
    MOVE   186 TO CURR-INDEX, CALL 'TDRP20'.
    ADD 001 TO CTR-1.
    MOVE TAB-JOB (CTR-1) TO PW-JOB7.
    MOVE   187 TO CURR-INDEX, CALL 'TDRP20'
    IF CTR-1 EQUALS TBLE-CTR
    MOVE   187 TO CURR-INDEX, CALL 'TDRP30'
```

Figure 2. Instrumented COBOL Source Program
(Continued)

```
     GO TO LAST-CLOS.
     MOVE   188 TO CURR-INDEX, CALL 'TDRP20'.
     ADD 001 TO CTR-1.
     MOVE TAB-JOB (CTR-1) TO PW-JOB8.
     MOVE   189 TO CURR-INDEX, CALL 'TDRP20'
     IF CTR-1 EQUALS TBLE-CTR
     MOVE   189 TO CURR-INDEX, CALL 'TDRP30'
     GO TO LAST-CLOS.
     MOVE   190 TO CURR-INDEX, CALL 'TDRP20'.
     ADD 001 TO CTR-1.
     MOVE TAB-JOB (CTR-1) to PW-JOB9.
     WRITE P-REC BEFORE ADVANCING 2 LINES.
     MOVE SPACES TO P-REC.
     ADD 02 TO LIN-CTR.
     GO TO WR-WHIP.
 LAST-CLOS.
     MOVE   191 TO CURR-INDEX, CALL 'TDRP20'.
     WRITE P-REC BEFORE 01.
     CLOSE P-FILE.
     MOVE +1000 TO REC-CNT
     GO TO WRITE-TDRP-FILE.
 WHIP-HD.
     MOVE   192 TO CURR-INDEX, CALL 'TDRP20'.
     MOVE SPACES TO P-REC.
     WRITE P-REC BEFORE ADVANCING NEXT-PG.
     ADD 0001 TO PG-CTR.
     MOVE 'LISTING OF JOBS ON LINK NO/JOB ORDER FILE BUT NOT ON
-    'MAST' TO P-HD2-W.
     MOVE 'ER JOB ORDER FILE FOR ' TO P-HD2A-W.
     MOVE TODAYS-DT TO P-DT-W.
     MOVE 'PAGE ' TO P-W-PG.
     MOVE PG-CTR TO P-W-ID.
     WRITE P-REC BEFORE ADVANCING 2 LINES.
     MOVE SPACES TO P-REC.
     MOVE '  JOB-NO       JOB-NO       JOB-NO      JOB-NO      J
-    'OB  ' TO P-HD2.
     MOVE '-NO       JOB-NO       JOB-NO      JOB-NO      JOB-NO
-    '  ' TO P-HD2A.
     WRITE P-REC BEFORE ADVANCING 2 LINES.
     MOVE SPACES TO P-REC.
     MOVE 04 TO LIN-CTR.
```

Figure 2. Instrumented COBOL Source Program
(Continued)

```
             ** TDRP REFORMATTED REPORT PROGRAM **                 EXECUTION     TRUE     FALSE
                                                                     COUNT       PATH      PATH
CK-MSTR.
    IF IM-JOB EQUALS HOLD-JOB                                        57983       1485     56498
    GO TO EVEN-MAT.                                                   1485
    IF IM-JOB IS GREATER THAN HOLD-JOB                               56498         81     56417
    MOVE 1 TO CK-CD                                                     81
    GO TO                                                               81
    UNEVEN.
    MOVE IN-MSTR TO OT-MSTR.                                         56417
    IF OM-3 EQUALS 5 AND OM-4 IS LESS THAN TODAYS-DT                 56417          0     56417
    MOVE 3 TO                                                            0
    OM-3
    PERFORM PR-PUN.                                                      0
    IF OM-3 EQUALS 5 AND OM-5 IS LESS THAN TODAYS-DT                 56417          0     56417
    MOVE 4 TO                                                            0
    OM-3
    PERFORM PR-PUN.                                                      0
    IF OM-6 EQUALS 8 AND OM-7 IS LESS THAN TODAYS-DT                 56417          0     56417
    MOVE 7 TO                                                            0
    OM-6
    PERFORM PR-PUN.                                                      0
    IF OM-6 EQUALS 8 AND OM-8 IS LESS THAN TODAYS-DT                 56417          0     56417
    MOVE 9 TO                                                            0
    OM-6
    PERFORM PR-PUN.                                                      0
    IF OM-JOB IS UNEQUAL TO SPACES                                   56417      56417         0
    WRITE OT-MSTR                                                    56417
    ADD 1 TO MSTR-1100-CT.                                           56417
    GO TO CK-WHIP.                                                   56417
EVEN-MAT.
    MOVE IN-MSTR TO OT-MSTR.                                          1485
    MOVE 3 TO CK-CD.                                                  1485
UNEVEN.
    IF H-ONE IS UNEQUAL TO SPACES                                     1566        495      1071
    PERFORM MOV-H1.                                                    495
    IF H-TWO IS UNEQUAL TO SPACES                                     1566        516      1050
    PERFORM MOV-H2.                                                    516
    IF H-THREE IS UNEQUAL TO SPACES                                   1566        115      1451
    PERFORM MOV-H3 THRU EX-1.                                          115
    IF H-FOUR IS UNEQUAL TO SPACES                                    1566        497      1069
    PERFORM MOV-H4 THRU EX-1.                                          497
    IF OM-JOB IS UNEQUAL TO SPACES                                    1566       1566         0
    PERFORM PR-PUN.                                                   1566
    IF OM-J1B EQUALS 3                                                1566        564      1002
    MOVE OM-J1C TO OM-36A                                              564
    MOVE OM-J1A TO                                                     564
    OM-36B
    ELSE
    MOVE OM-JOB TO OM-36.                                             1002
    MOVE SPACES TO H-ONE H-TWO H-THREE H-FOUR.                        1566
    IF OM-JOB IS UNEQUAL TO SPACES                                    1566       1566         0
    WRITE OT-MSTR                                                     1566
    ADD 1 TO MSTR-1100-CT.                                            1566
CK-WHIP.
    IF OM-JOB EQUALS HOLD-W-JOB                                      62018      15166     46852
    GO TO MAT-WHIP.                                                  15166
    IF HOLD-W-JOB IS LESS THAN OM-JOB                                46852       4035     42817
    GO TO LO-WHIP.                                                    4035
    MOVE SPACES TO OT-MSTR.                                          42817
    IF CK-CD EQUALS 1                                                42817          0     42817
    GO TO MOV1.                                                          0
    IF CK-CD EQUALS 3                                                42817        661     42156
    ALTER 1-SW                                                         661
    TO PROCEED TO MOV1
```

Figure 3. TDRP Output Report

```
            ** TDRP REFORMATTED REPORT PROGRAM **                         EXECUTION    TRUE     FALSE
                                                                            COUNT      PATH     PATH

    ELSE
    ALTER 1-SW                                                              42156
    TO PROCEED TO CK-MSTR.
    MOVE 0 TO CK-CD.                                                        42817
    GO TO READ-M.                                                           42817
MAT-WHIP.
    MOVE SPACES TO OT-MSTR.                                                 15166
    IF CK-CD EQUALS 1                                                       15166       81    15085
    ALTER 2-SW                                                                 81
    TO PROCEED TO MOV1
    GO TO                                                                      81
    WHIP-MOV.
    IF CK-CD EQUALS 3                                                       15085      824    14261
    ALTER 2-SW                                                                824
    TO PROCEED TO MOV1
    ELSE
    ALTER 2-SW                                                              14261
    TO PROCEED TO CK-MSTR.
    ALTER 1-SW                                                              15085
    TO PROCEED TO WHIP-MOV.
    MOVE ZERO TO CK-CD.                                                     15085
    GO TO READ-M.                                                           15085
LO-WHIP.
    IF TBLE-CTR IS LESS THAN 999                                             4035      999     3036
    ADD 001 TO TBLE-CTR.                                                      999
    MOVE HOLD-W-JOB TO TAB-JOB (TBLE-CTR).                                   4035
LO-W-RTN.
    ALTER 2-SW                                                               4035
    TO PROCEED TO CK-WHIP.
    GO TO WHIP-MOV.                                                          4035
CLOS-MSTR.
    IF S-REC EQUALS HIGH-VALUES AND CR-IN EQUALS HIGH-VALUES                    1        1        0
    GO                                                                          1
    TO FIN-CL.
    MOVE HIGH-VALUES TO IN-MSTR.                                                0
    GO TO 1-SW.                                                                 0
CLOS-WHIP.
    IF S-REC EQUALS HIGH-VALUES AND IN-MSTR EQUALS HIGH-VALUES                  1        0        1
    GO TO FIN-CL.                                                               0
    MOVE HIGH-VALUES TO CR-IN.                                                  1
    GO TO 1A-SW.                                                                1
CLOS-DET.
    IF IN-MSTR EQUALS HIGH-VALUES AND CR-IN EQUALS HIGH-VALUES                  1        0        1
    GO TO FIN-CL-DET.                                                           0
    MOVE HIGH-VALUES TO S-REC.                                                  1
    GO TO 3-SW.                                                                 1
FIN-CL-DET.
    IF H-ONE EQUALS SPACES AND H-TWO EQUALS SPACES AND H-THREE                  0        0        0
    EQUALS SPACES AND H-FOUR EQUALS SPACES
    GO TO FIN-CL.                                                               0
    IF H-ONE EQUALS HIGH-VALUES                                                 0        0        0
    GO TO FIN-CL.                                                               0
    PERFORM UNEVEN.                                                             0
FIN-CL.
    CLOSE MSTR-IN CR-FILE MSTR-OT PUNCH-OT.                                     1
    DISPLAY '1100 MASTER RCDS ' MSTR-1100-CT UPON PRINTER.                      1
    MOVE 99 TO LIN-CTR.                                                         1
    MOVE ZEROES TO PG-CTR.                                                      1
WR-WHIP.
    IF CTR-1 EQUALS TBLE-CTR                                                  112        1      111
    GO TO LAST-CLOS.                                                            1
```

Figure 3. TDRP Output Report
(Continued)

```
                  ** TDRP REFORMATTED REPORT PROGRAM **        EXECUTION    TRUE     FALSE
                                                                 COUNT      PATH      PATH
     IF LIN-CTR IS GREATER THAN 50                                     111        5       106
     PERFORM WHIP-HD.                                                    5
     ADD 001 TO CTR-1.                                                 111
     MOVE TAB-JOB (CTR-1) TO PW-JOB1.                                  111
     IF CTR-1 EQUALS TBLE-CTR                                          111        0       111
     GO TO LAST-CLOS.                                                    0
     ADD 001 TO CTR-1.                                                 111
     MOVE TAB-JOB (CTR-1) TO PW-JOB2.                                  111
     IF CTR-1 EQUALS TBLE-CTR                                          111        0       111
     GO TO LAST-CLOS.                                                    0
     ADD 001 TO CTR-1.                                                 111
     MOVE TAB-JOB (CTR-1) TO PW-JOB3.                                  111
     IF CTR-1 EQUALS TBLE-CTR                                          111        0       111
     GO TO LAST-CLOS.                                                    0
     ADD 001 TO CTR-1.                                                 111
     MOVE TAB-JOB (CTR-1) TO PW-JOB4.                                  111
     IF CTR-1 EQUALS TBLE-CTR                                          111        0       111
     GO TO LAST-CLOS.                                                    0
     ADD 001 TO CTR-1.                                                 111
     MOVE TAB-JOB (CTR-1) TO PW-JOB5.                                  111
     IF CTR-1 EQUALS TBLE-CTR                                          111        0       111
     GO TO LAST-CLOS.                                                    0
     ADD 001 TO CTR-1.                                                 111
     MOVE TAB-JOB (CTR-1) TO PW-JOB6.                                  111
     IF CTR-1 EQUALS TBLE-CTR                                          111        0       111
     GO TO LAST-CLOS.                                                    0
     ADD 001 TO CTR-1.                                                 111
     MOVE TAB-JOB (CTR-1) TO PW-JOB7.                                  111
     IF CTR-1 EQUALS TBLE-CTR                                          111        0       111
     GO TO LAST-CLOS.                                                    0
     ADD 001 TO CTR-1.                                                 111
     MOVE TAB-JOB (CTR-1) TO PW-JOB8.                                  111
     IF CTR-1 EQUALS TBLE-CTR                                          111        0       111
     GO TO LAST-CLOS.                                                    0
     ADD 001 TO CTR-1.                                                 111
     MOVE TAB-JOB (CTR-1) TO PW-JOB9.                                  111
     WRITE P-REC BEFORE ADVANCING 2 LINES.                             111
     MOVE SPACES TO P-REC.                                             111
     ADD 02 TO LIN-CTR.                                                111
     GO TO WR-WHIP.                                                    111
 LAST-CLOS.
     WRITE P-REC BEFORE 01.                                              1
     CLOSE P-FILE.                                                       1
     STOP RUN.                                                           1
 WHIP-HD.
     MOVE SPACES TO P-REC.                                               5
     WRITE P-REC BEFORE ADVANCING NEXT-PG.                               5
     ADD 0001 TO PG-CTR.                                                 5
     MOVE 'LISTING OF JOBS ON LINK NO/JOB ORDER FILE BUT NOT ON          5
-    'MAST' TO P-HD2-W.
     MOVE 'ER JOB ORDER FILE FOR ' TO P-HD2A-W.                          5
     MOVE TODAYS-DT TO P-DT-W.                                           5
     MOVE 'PAGE ' TO P-W-PG.                                             5
     MOVE PG-CTR TO P-W-ID.                                              5
     WRITE P-REC BEFORE ADVANCING 2 LINES.                               5
     MOVE SPACES TO P-REC.                                               5
     MOVE '  JOB-NO       JOB-NO       JOB-NO     JOB-NO       J         5
-    'OB  ' TO P-HD2.
     MOVE '-NO       JOB-NO       JOB-NO     JOB-NO       JOB-NO         5
-    '  ' TO P-HD2A.
     WRITE P-REC BEFORE ADVANCING 2 LINES.                               5
     MOVE SPACE TO P-REC.                                                5
     MOVE 04 TO LIN-CTR.                                                 5
```

Figure 3. TDRP Output Report
(Continued)

Compile and execution times will be affected by the code insertions. The increase in compile and execution times for three sample test cases are evident in Figure 4.

COMPILE TIMES IN SECONDS

	BEFORE	AFTER	% INCREASE
TEST 1	18.74	26.7	42
TEST 2	27.71	39.4	42
TEST 3	33.41	40.4	21

EXECUTION TIMES IN SECONDS

	BEFORE	AFTER	% INCREASE
TEST 1	1998.4	3010.3	51
TEST 2	292.2	393.7	34
TEST 3	824.9	993.9	20

Figure 4. Sample compile and execution times

The additional memory requirement for execution is less than 2K words which include object modules and data buffers, but not inserted COBOL statements.

Lastly, only one data file may be reduced per TDRP run. All original files must be used in a TDRP run even though reduced files may exist to ensure that all logic paths executed for the original data file are exercised. In addition, if a reduced file is run through TDRP to be reduced a second time, tests have shown testing coverage is affected. It should be noted that the same level of testing coverage is uncertain when several reduced files are used in testing since the original intention of TDRP was to reduce a single master file to conserve computer resources over multiple runs (2).

6. Examples

The following procedure was used as a guideline for each test case run. Initially, the program was run with the original data base. Then the TDRP run was made producing the reduced data base. A ninety percent reduction in the original data base was user-specified. Lastly, the program was run with the reduced data base to ascertain the savings in CPU time.

In the first test (Figure 5), two master files were reduced. The savings in storage and CPU time become quite apparent when both reduced files are used to run a test.

TEST 1

PA-MSTR FILE

	ORIGINAL DATA FILE	AFTER REDUCTION	PERCENTAGE SAVINGS
NUMBER OF RECORDS (STORAGE)	45450	4548	89.99
CPU TIME (SECONDS)	757.71	670.72	11.48

DU-MSTR FILE

	ORIGINAL DATA FILE	AFTER REDUCTION	PERCENTAGE SAVINGS
NUMBER OF RECORDS (STORAGE)	28670	2868	90.00
CPU TIME (SECONDS)	757.71	245.16	67.64

COMBINED PA-MSTR AND DU-MSTR FILES

	ORIGINAL DATA FILES	AFTER REDUCTION	PERCENTAGE SAVINGS
TOTAL NO. OF RECORDS (STORAGE)	74120	7416	89.99
CPU TIME (SECONDS)	757.71	157.49	79.22

Figure 5. Results from Test 1

In tests 2 and 3 (Figure 6), only one master file was being processed. In both test cases, the savings in CPU time are considerate.

TEST 2

PW115M01-IN

	ORIGINAL DATA FILE	AFTER REDUCTION	PERCENTAGE SAVINGS
NUMBER OF RECORDS (STORAGE)	13021	1314	89.91
CPU TIME (SECONDS)	98.29	16.04	83.68

TEST 3

PWJ43D01-IN

	ORIGINAL DATA FILE	AFTER REDUCTION	PERCENTAGE SAVINGS
NUMBER OF RECORDS (STORAGE)	13829	1386	89.98
CPU TIME (SECONDS)	257.86	54.08	79.03

Figure 6. Results from Tests 2 and 3

7. Constraints

TDRP currently handles up to 1000 conditional statements but may easily be modified to handle more. Files for reduction should not be used in a COBOL sort since the program needs to instrument the "read" statement for a file in order to extract the records. Lastly, at present, data files with control or dependent records will yield unpredictable results after reduction. This control record problem has been analyzed and modifications are currently being unit tested.

References

(1) Basili, V. and Perricone, B., "Software Errors and Complexity: An Empirical Investigation", Proceedings of the Seventh Annual Software Engineering Workshop, December, 1982, pp. 1-49.

(2) Navy Regional Data Automation Center (NARDAC) Washington, D.C., "Test Data Reduction Program (TDRP) for UNIVAC 1100 Series Computers Users Manual," NARDAC Washington, DC Document No. UK50026S, UM-02, May 1982.

NBS-114A (REV. 2-80)

U.S. DEPT. OF COMM. **BIBLIOGRAPHIC DATA SHEET** *(See instructions)*	1. PUBLICATION OR REPORT NO. NBS SP 500-104	2. Performing Organ. Report No.	3. Publication Date October 1983

4. TITLE AND SUBTITLE

Computer Science and Technology:
Proceedings of the Nineteenth Meeting of the Computer Performance Evaluation Users Group (CPEUG)

5. AUTHOR(S)

Deborah Mobray, Editor

6. PERFORMING ORGANIZATION *(If joint or other than NBS, see instructions)*

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

7. Contract/Grant No.

8. Type of Report & Period Covered

Final

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS *(Street, City, State, ZIP)*

Same as No. 6

10. SUPPLEMENTARY NOTES

Library of Congress Catalog Card Number: 83-600594

□ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT *(A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)*

These Proceedings record the papers that were presented at the Nineteenth Meeting of the Computer Performance Evaluation Users Group (CPEUG 83) held October 25-28, 1983 in San Francisco, CA. CPEUG 83 recognized the rapid introduction of sophisticated end-user technology into the information processing environment and addressed the challenges posed to the CPE community. CPEUG 83 offered topics ranging from microcomputers to supercomputers. The increasingly complex area of data communications was presented as well as topics in office automation, software improvement and engineering, capacity planning, and quality assurance. The program was divided into three parallel sessions and included technical papers, case studies, tutorials, and panels. Technical papers are presented in the Proceedings in their entirety.

12. KEY WORDS *(Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)*

acquisition; benchmarking; capacity planning; cost accounting and chargeback; data communications; end-user computing; local area networks; microcomputers; modeling techniques; office automation; software engineering.

13. AVAILABILITY

☒ Unlimited
□ For Official Distribution. Do Not Release to NTIS
☒ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.
□ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

235

15. Price

$6.50

ANNOUNCEMENT OF NEW PUBLICATIONS ON COMPUTER SCIENCE & TECHNOLOGY

Superintendent of Documents,
Government Printing Office,
Washington, DC 20402

Dear Sir:

Please add my name to the announcement list of new publications to be issued in the series: National Bureau of Standards Special Publication 500-.

Name ____

Company ____

Address ____

City ____________________ State ____________ Zip Code ________

(Notification key N-503)

☆ U.S. GOVERNMENT PRINTING OFFICE : 1983 O—380-997 (5207)

NBS TECHNICAL PUBLICATIONS

PERIODICALS

JOURNAL OF RESEARCH—The Journal of Research of the National Bureau of Standards reports NBS research and development in those disciplines of the physical and engineering sciences in which the Bureau is active. These include physics, chemistry, engineering, mathematics, and computer sciences. Papers cover a broad range of subjects, with major emphasis on measurement methodology and the basic technology underlying standardization. Also included from time to time are survey articles on topics closely related to the Bureau's technical and scientific programs. As a special service to subscribers each issue contains complete citations to all recent Bureau publications in both NBS and non-NBS media. Issued six times a year. Annual subscription: domestic $18; foreign $22.50. Single copy, $5.50 domestic; $6.90 foreign.

NONPERIODICALS

Monographs—Major contributions to the technical literature on various subjects related to the Bureau's scientific and technical activities.

Handbooks—Recommended codes of engineering and industrial practice (including safety codes) developed in cooperation with interested industries, professional organizations, and regulatory bodies.

Special Publications—Include proceedings of conferences sponsored by NBS, NBS annual reports, and other special publications appropriate to this grouping such as wall charts, pocket cards, and bibliographies.

Applied Mathematics Series—Mathematical tables, manuals, and studies of special interest to physicists, engineers, chemists, biologists, mathematicians, computer programmers, and others engaged in scientific and technical work.

National Standard Reference Data Series—Provides quantitative data on the physical and chemical properties of materials, compiled from the world's literature and critically evaluated. Developed under a worldwide program coordinated by NBS under the authority of the National Standard Data Act (Public Law 90-396).

NOTE: The principal publication outlet for the foregoing data is the Journal of Physical and Chemical Reference Data (JPCRD) published quarterly for NBS by the American Chemical Society (ACS) and the American Institute of Physics (AIP). Subscriptions, reprints, and supplements available from ACS, 1155 Sixteenth St., NW, Washington, DC 20056.

Building Science Series—Disseminates technical information developed at the Bureau on building materials, components, systems, and whole structures. The series presents research results, test methods, and performance criteria related to the structural and environmental functions and the durability and safety characteristics of building elements and systems.

Technical Notes—Studies or reports which are complete in themselves but restrictive in their treatment of a subject. Analogous to monographs but not so comprehensive in scope or definitive in treatment of the subject area. Often serve as a vehicle for final reports of work performed at NBS under the sponsorship of other government agencies.

Voluntary Product Standards—Developed under procedures published by the Department of Commerce in Part 10, Title 15, of the Code of Federal Regulations. The standards establish nationally recognized requirements for products, and provide all concerned interests with a basis for common understanding of the characteristics of the products. NBS administers this program as a supplement to the activities of the private sector standardizing organizations.

Consumer Information Series—Practical information, based on NBS research and experience, covering areas of interest to the consumer. Easily understandable language and illustrations provide useful background knowledge for shopping in today's technological marketplace.

Order the **above** *NBS publications from: Superintendent of Documents, Government Printing Office, Washington, DC 20402.*

Order the **following** *NBS publications—FIPS and NBSIR's—from the National Technical Information Service, Springfield, VA 22161.*

Federal Information Processing Standards Publications (FIPS PUB)—Publications in this series collectively constitute the Federal Information Processing Standards Register. The Register serves as the official source of information in the Federal Government regarding standards issued by NBS pursuant to the Federal Property and Administrative Services Act of 1949 as amended, Public Law 89-306 (79 Stat. 1127), and as implemented by Executive Order 11717 (38 FR 12315, dated May 11, 1973) and Part 6 of Title 15 CFR (Code of Federal Regulations).

NBS Interagency Reports (NBSIR)—A special series of interim or final reports on work performed by NBS for outside sponsors (both government and non-government). In general, initial distribution is handled by the sponsor; public distribution is by the National Technical Information Service , Springfield, VA 22161, in paper copy or microfiche form.