

117996

~~21572~~

---

BY THE COMPTROLLER GENERAL

# Report To The Congress

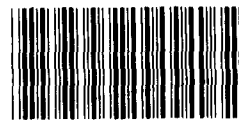
OF THE UNITED STATES

---

## Improving Cobol Applications Can Recover Significant Computer Resources

Computer applications written in the COBOL programming language are used to do a large portion of Federal data processing and consume a large portion of Federal computer resources. Such applications can sometimes be improved to do the job with fewer computer resources, thus freeing significant resources for other jobs, and perhaps deferring computer replacement.

The National Bureau of Standards should issue guidance on the use of COBOL for applications, and agency ADP management should review applications periodically for possible improvements.



117996

AFMD-82-4  
APRIL 1, 1982

0 21272

**Request for copies of GAO reports should be sent to:**

**U.S. General Accounting Office  
Document Handling and Information  
Services Facility  
P.O. Box 6015  
Gaithersburg, Md. 20760**

**Telephone (202) 275-6241**

**The first five copies of individual reports are free of charge. Additional copies of bound audit reports are \$3.25 each. Additional copies of unbound report (i.e., letter reports) and most other publications are \$1.00 each. There will be a 25% discount on all orders for 100 or more copies mailed to a single address. Sales orders must be prepaid on a cash, check, or money order basis. Check should be made out to the "Superintendent of Documents".**



COMPTROLLER GENERAL OF THE UNITED STATES  
WASHINGTON D.C. 20548

B-206179

To the President of the Senate and the  
Speaker of the House of Representatives

Computer software is the most important part of automatic data processing systems today. It is expensive to develop and maintain, and poorly developed software wastes computer resources in operation.

This report discusses reducing the computer costs of Federal computer applications and includes a checklist to guide ADP managers in such work. We made this review because we wanted to explore the management implications and economics of improving applications written in COBOL--the most widely used Federal programming language.

We are sending copies of this report to the Secretary of Commerce and to the Administrator of General Services.

*Charles A. Brooker*

Comptroller General  
of the United States





D I G E S T

Federal use of computers and computer programs is extensive. COBOL--the Common Business Oriented Language--is the most widespread programming language used for Federal business applications. Business applications are automated with computer programs written in COBOL, with files, and with a computing environment in which they are operated. COBOL applications consume machine resources in processing data and reporting and storing results. The extent to which COBOL applications consume machine resources depends upon how they are written and the environment in which they operate.

This report addresses opportunities for savings by reducing the resources consumed by existing COBOL applications (see ch. 2) but warns that efforts to reduce machine resource consumption must be carefully managed, so as not to conflict with other automatic data processing (ADP) management objectives. (See ch. 3.)

The roles of agencies include (1) Government-wide roles for the General Services Administration (GSA), the Commerce Department, and the Office of Management and Budget (OMB), specified in the Brooks Act (Public Law 89-306), (2) the augmented role of OMB specified in the Paperwork Reduction Act (Public Law 96-511), (3) the management responsibilities of Federal agencies specified in OMB Circular A-71, and (4) GAO concern with ADP management in its role of aiding the Congress. In particular, the Institute for Computer Science and Technology, Department of Commerce, has the mission of providing Federal agencies with ADP technological advisory services. In this review, GAO combined tests of its own, results reported by other agencies, discussions with experts, and review of current literature to review the management, economics, potential benefits, and applicability of techniques to reduce the machine resources consumed by COBOL applications.

THE COMPUTER RESOURCES COBOL  
APPLICATIONS CONSUME CAN  
BE REDUCED SIGNIFICANTLY

Significant benefits have been achieved at some Federal installations by reducing the machine resources consumed by COBOL applications. For example, the Department of Housing and Urban Development improved five applications and, in the first year alone, recovered computer time valued at \$83,400 in return for a \$19,000 cost of doing the work--about a 4 to 1 return. They estimated that the improved applications will run about 5 years, which would accumulate a savings of about \$400,000--about a 20 to 1 pay-back. (See p. 9.) While such returns cannot be realized from all COBOL applications (see ch. 3) we feel that they show a significant potential for savings. Many other installations, however, have done little or nothing in this area.

The National Bureau of Standards (NBS) has not published specific guidance on the effective and efficient use of COBOL for applications, even though it has a technical guidance mission. The benefits of such work can be measured and verified, and such work can be done on any brand of computer using COBOL. A systematic method and the use of automated tools can increase the pay-off. COBOL applications can also be newly developed with deliberate attention to using machine resources efficiently. If properly managed, both types of efforts can have an excellent pay-off. In particular, we feel that training programmers to write new programs better has a large potential for improving COBOL applications.

EFFORTS TO REDUCE COMPUTER RESOURCE  
CONSUMPTION ARE NOT ALWAYS APPROPRIATE  
AND MUST BE CAREFULLY CONSIDERED

While significant benefits can be achieved by modifying COBOL applications to reduce their machine resource consumption, such work must be considered in light of other ADP management objectives. For example, satisfying users' application needs may take precedence over cost reduction on the current computers. Work done to reduce machine resource consumption should not violate other software management objectives. Efforts to reduce COBOL applications' consumption of machine resources may not require continuous effort at individual sites, and some

applications are not worth the effort it would take to reduce their machine costs. Thus, prudent management of this type of work is needed if good results are to be realized.

### CONCLUSIONS

Significant computer resources can be recovered for other purposes when work to reduce their consumption by COBOL applications is properly done. The potential aggregate benefit to the Federal Government is quite large because of the widespread use of COBOL and the long life of many COBOL programs. More needs to be done to raise ADP managers' and users' concern with the cost of applications, and to increase programmers' efficiency and effectiveness in building and maintaining COBOL applications. Agencies with Government-wide ADP responsibilities should publish guidance on reducing machine resources consumed by COBOL applications. GAO believes that COBOL applications deserve separate treatment because of the extensive Federal use of COBOL.

The general methods of managing and doing work to reduce COBOL applications' consumption of machine resources can be used on any brand of computers that implements COBOL.

Work should be done to improve existing applications only when the value of the recovered resources will significantly exceed the cost of doing the work and the work will not harm higher priority objectives. Many of the techniques used are similar from one brand of computer to another and, once learned, are relatively easy to apply. Efficient machine resource usage can also be built into new COBOL applications. We feel that building new applications better offers more potential than reworking old ones.

Efforts to reduce machine resources consumed by COBOL applications need not violate other software management objectives such as maintainability, ease of conversion, and adherence to language standards.

### RECOMMENDATIONS

The Secretary of Commerce should direct NBS to publish guidance on the effective and efficient use of COBOL for applications. GAO believes the guidance should include examples taken from real

life applications and that a possible starting point would be to use a table of contents similar to that of the already published USING ANSI FORTRAN 1/ and the material in GAO's provisional checklist (app. I). GAO also believes that GSA's Office of Software Development and the Federal Computer Performance Evaluation and Simulation Center (FEDSIM) could work with NBS in constructing such guidance.

Heads of Federal agencies should require periodic review of the machine resource consumption of COBOL applications at their installations, and, where appropriate, require action to reduce the consumption of the expensive applications.

#### AGENCY COMMENTS

GAO asked for comments from the Department of Commerce, GSA, and FEDSIM. They all furnished comments, which are included verbatim in this report as appendix IV. The sites GAO visited were all allowed to review and discuss summaries soon after the visits.

GSA (1) said its Office of Software Development would follow GAO's recommendation to work with NBS in publishing guidance on the effective and efficient use of COBOL, (2) suggested further emphasis on testing, and (3) pointed to a document of its own on software improvement. These comments resulted in increased coverage of testing in GAO's provisional checklist and the addition of GSA's document to the list of references. (See pp. 66-68.)

NBS stated that this report is a worthwhile attempt to focus on the need to pay careful and continuous attention at the ADP installation level to the efficiency of application software. NBS did not, however, concur with GAO's recommendation that it publish guidance on the effective and efficient use of COBOL for applications. (See p. 63.) GAO believes that specific guidance on COBOL is worthwhile because of the extensive Federal use of COBOL and that GSA may publish the guidance if NBS does not.

FEDSIM had no comment on the substance of the draft report but asked that GAO clarify that the person spoken to at FEDSIM did not express a FEDSIM position. (See p. 69.)

---

1/NBS Handbook 131.

# C o n t e n t s

		<u>Page</u>
DIGEST		i
CHAPTER		
1	INTRODUCTION	1
	Federal use of computers and software is extensive	1
	Types of computer software	1
	COBOL is the most widespread Federal programming language for business applications	4
	Machine resource consumption of operating COBOL applications depends on how they are written and on their environment	5
	Roles of various agencies	5
	Objectives, scope, and methodology	6
2	SIGNIFICANT BENEFITS CAN BE ACHIEVED BY REDUCING MACHINE RESOURCES CONSUMED BY COBOL APPLICATIONS	8
	Some Federal installations have achieved significant benefits by optimizing COBOL applications to reduce machine resource consumption	8
	Some Federal installations have done little or nothing to reduce the machine costs of their applications	12
	The benefits can be measured and verified	13
	A systematic approach helps improvement efforts	14
	Benefits are potentially widespread and available on any brand of computer that has COBOL	15
	Automated tools can help improvement efforts	16
	Applications can be developed to deliberately require less machine resources	18
3	EFFORTS TO REDUCE MACHINE RESOURCES USED BY COBOL APPLICATIONS ARE NOT ALWAYS APPROPRIATE AND MUST BE CAREFULLY CONSIDERED	19
	Other ADP management objectives may have higher priority than attempts to reduce machine resource consumption	19

	<u>Page</u>	
Work done to reduce machine costs should not violate other software management objectives	20	
Continual efforts to reduce COBOL applications' machine resource consumption may not be cost effective	22	
Conclusions, recommendations, and agency comments	24	
APPENDIX		
I	Provisional checklist for reducing the machine resources consumed by COBOL applications	27
II	Summaries of what we found at the sites visited	46
III	List of references	58
IV	Official Comments from the Department of Commerce, General Services Administration, and Federal Computer Performance Evaluation and Simulation Center	62

#### ABBREVIATIONS

ADP	automatic data processing
COBOL	Common Business Oriented Language
CPU	central processing unit
FEDSIM	Federal Computer Performance Evaluation and Simulation Center
GSA	General Services Administration
HUD	Housing and Urban Development
NBS	National Bureau of Standards
OMB	Office of Management and Budget
USACSC	U.S. Army Computer Systems Command
USAMSSA	U.S. Army Management Systems Support Agency

## CHAPTER 1

### INTRODUCTION

Federal use of computers and computer programs is extensive. In fact, the Federal Government is the world's largest user of automatic data processing (ADP) resources. COBOL--the Common Business Oriented Language--is the most widespread programming language used for and by the Federal Government. And it is the optimum use of machine resources for applications whose programs are written in COBOL that we will be concerned with in this report.

All Federal agencies are involved at some level in the role of the computer in the Government. The General Services Administration (GSA) and the National Bureau of Standards (NBS) have Government-wide responsibilities, as provided under Public Law 89-306, often called the Brooks Act. The Paperwork Reduction Act (Public Law 96-511) specifies the Government-wide role of OMB. The management responsibilities of other Federal agencies are specified in Office of Management and Budget (OMB) Circular A-71, and we are concerned with automatic data processing (ADP) management in our role of aiding the Congress.

#### FEDERAL USE OF COMPUTERS AND SOFTWARE IS EXTENSIVE

As the world's largest user of ADP resources, the Federal Government incurs costs that have been estimated at over \$15 billion per year and which continue to increase. The General Services Administration's ADP inventory for fiscal 1980 reported that the Government owns or leases over 15,100 computers. These computers are used to process a variety of applications ranging from delivering health and welfare services, to administering social security and veterans' benefits, to exploring space, to analyzing and reporting on military matters such as readiness levels.

Computer programs--generally called "software" in the industry--make all these computers run. A computer without programs is like a phonograph without records--it won't play.

#### TYPES OF COMPUTER SOFTWARE

Computer software has been defined as the programs that make the computer run, the data files that those programs process, and explanatory material--called documentation--that accompanies the programs and files. Often, however, the word "software" is used to refer only to the programs, which can generally be grouped into systems programs, utility programs, and applications programs.

##### Systems programs

Systems programs automate the control of operation of the computer and auxiliary equipment. They control the running of

utility programs and applications programs, control the allocation of machine resources to the programs, and report on the resources used to run the programs. Systems programs are usually supplied by the computer vendor but may be obtained from other suppliers.

### Utility programs

Utility programs aid the tasks of computer programmers and others who work with the computer. They include language translators <sup>1/</sup> and stored routines for very common tasks, such as sorting data. Utility programs may be supplied by the hardware vendor or independent software firms or may be written by the user's employees. Software tools are a particular class of utility programs which aid work done on other computer programs.

### Applications programs

These programs automate the tasks of end users. For example, an applications program of a payroll system is the checkwriting program which writes checks for employees. The tasks automated for end users are almost endless: payroll, billing, and inventory in the business sector; simulations and statistical processing in the scientific sector; and air traffic control and satellite tracking in the command, control, and communications sector.

Applications programs have life cycles which can be divided into a development phase and an operational or production phase. The development phase consists of defining the requirements, designing the application, programming, and testing. The objective of applications software development is to construct computer programs that will process the users' data correctly at as low a cost as feasible and to document those programs so they can easily be modified later if necessary. Software development is a labor-intensive, error-prone process. Errors can be made both in deciding what the programs should do and in constructing them to do it.

The operational phase begins when the application produces its first user output. The phase includes:

- Maintenance. Work done on the programs and/or their documentation after they are in production to correct errors and omissions. Many people use the term "maintenance" to include the work we call "modification" below.
- Modification. Work done to make an existing system accomplish additional user requirements above and beyond those originally intended.

---

<sup>1/</sup>Language translators are compilers and interpreters that transform the statements of programming languages written by humans into internal machine codes which directly control computers.



- Performance improvement. Work done on operational applications to make them consume fewer resources in operation. It is also called optimization.
- Conversion. Work done to make programs run on a computer other than the one for which they were originally written. This work may be done during the operational phase of the life of applications software.

The work done on programs and documentation during their operational phase can account for up to 70 percent of total life cycle costs and consume most of an organization's computer programming labor. Experience has shown that continual changes to computer programs have often been necessary to (1) correct hidden errors, (2) add new user functions because of legal, administrative, or technical changes, and (3) reduce the machine resources they consume. The objectives of software maintenance and modification are to fix errors as soon as possible and to install needed user changes correctly with as little effort as feasible.

The objective of performance improvement, or optimization, is to make changes to applications--programs, their files, and their environments 1/--to reduce the machine resources those applications consume. These changes are valid only when such reduction will repay its own cost without introducing errors into the logic that processes the user's needs and without undue conflict with other software management objectives. Optimization may be needed on the present hardware, for example, to defer the procurement of more hardware or may be needed after programs have been converted to new hardware. The latter is reported in our report on computer software conversion. 2/

Applications may be developed, maintained, modified, optimized, or converted to new hardware by employees of the user's organization, or these activities may be done by specialist firms. Programs may also be bought readymade from firms selling software built to serve many users. In some cases, firms that sell computers also will sell applications software.

### An illustration

Definitions are not standard in the software field--especially as to where systems programs end and utility programs begin. However, an example will illustrate our use of the terms. The

---

1/The program's environment includes the organization of its files on storage devices and its interfaces with the machine's operating system and with other programs.

2/"Conversion: A Costly Disruptive Process That Must Be Considered When Buying Computers," FGMSD-80-35, June 3, 1980.

checkwriting program of a payroll system is an applications program which prints checks. While the checkwriting program is running, its operation and its consumption of and access to computer resources such as memory, disk, and tape, are controlled by the supervisory control module of the operating system--a systems program. The checkwriting program would be written with the aid of utility programs, including a language translator. Also, the timecards containing the data processed by the checkwriting program may have been presorted for processing by another utility program--the sort/merge. The employee master records containing such information as name and pay rate will be stored on a storage resource, such as a tape.

COBOL IS THE MOST WIDESPREAD  
FEDERAL PROGRAMMING LANGUAGE  
FOR BUSINESS APPLICATIONS

The COBOL programming language is a Federal, national, and international standard. It was developed with Department of Defense (DOD) sponsorship in 1959 and 1960 and has been revised since.

Federal Information Processing Standards Publication 21-1, "COBOL," (FIPS PUB 21-1) says:

" \* \* \* the general intent of this publication is to provide a standard language that can be used in programming information processing applications except in circumstances, discussed below, where such use would not be advantageous."

GSA, which operates the Compiler Testing Center as part of its Office of Software Development (OSD), is now responsible for ensuring that COBOL compilers offered to the Government comply with the Federal COBOL standard.

In our recent review of software maintenance, 1/ 263 of 409 Federal installations responding to a questionnaire reported COBOL as their dominant application language. These 263 had an average inventory of 746 programs, and 170 of the installations reported they had over 100 COBOL programs in production. The 263 installations said that their COBOL programs last an average of 5.4 years. These figures were significantly higher than those reported for FORTRAN, 2/ the second most commonly reported language. FORTRAN is used at 212 installations for an average of 260 programs that last an average of 4.8 years.

---

1/"Federal Agencies' Maintenance of Computer Programs: Expensive and Undermanaged," AFMD-81-25, Feb. 26, 1981.

2/Formula Translator, a language developed for scientific and engineering applications.

MACHINE RESOURCE CONSUMPTION OF  
OPERATING COBOL APPLICATIONS  
DEPENDS ON HOW THEY ARE WRITTEN  
AND ON THEIR ENVIRONMENT

A COBOL application includes the COBOL program or programs, the files they process, and the environment in which the programs must operate and the files must be kept.

In writing COBOL programs, selecting the organization of the files they will process, and selecting other environmental factors, a programmer has many alternatives, many combinations of which will yield the same answers for the user. 1/ If better methods are chosen for writing the programs and arranging their interactions with their environment, the application will consume fewer machine resources in operation.

ROLES OF VARIOUS AGENCIES

The basic law governing Federal ADP management is the Brooks Act, Public Law 89-306. Under this act, the General Services Administration is responsible for procuring and maintaining Federal ADP resources. GSA receives technical advice from the Secretary of Commerce, primarily through the National Bureau of Standards, and both of these agencies get fiscal and policy guidance from the Office of Management and Budget. The role of OMB was further specified in the Paperwork Reduction Act (Public Law 96-511) which says that the Federal automatic data processing and telecommunications functions of the Director of OMB shall include developing and implementing policies, principles, standards, and guidelines for ADP and telecommunications functions of the Federal Government. NBS is responsible for providing scientific and technological advisory services to Federal agencies, for developing Federal Information Processing Standards, and for publishing guidance. The Federal Computer Performance Evaluation and Simulation Center (FEDSIM) was established to develop techniques for analyzing ADP systems for Federal agencies to improve utilization and performance.

In addition, each Federal agency has certain responsibilities for managing its own ADP resources. OMB Circular A-71, published in March 1965 by the Bureau of the Budget (now the Office of Management and Budget), states that the heads of all executive departments and establishments are responsible for the administration and management of their automatic data processing activities.

In our role of aiding the Congress, we are concerned with the management of Federal ADP and with computer software as an expensive part of Federal ADP. Our past reports to the Congress have

---

1/For example, calculate the pay correctly and print the correct amount on the paychecks.

recommended improvements in ADP management both on a Government-wide basis and to specific agencies. This COBOL review reflects our concern with effective and efficient Federal use of computer software and hardware resources.

#### OBJECTIVES, SCOPE, AND METHODOLOGY

In this review, we wanted to explore the management implications and economics of reducing machine resources consumed by COBOL applications and their applicability to different brands of computers, as well as specific tools and techniques for doing the work.

To cover these topics, our review included:

- Experiments of our own at five Federal sites to obtain data on what improvements could be achieved with various techniques and on the applicability of a general approach in different environments. (See app. II.)
- Review of the consumption of machine resources by COBOL programs at installations visited to verify the commonly reported phenomenon that a few programs consume a disproportionately large amount of total machine resources.
- Visits to organizations with continual optimization efforts (app. II, p. 53) to (1) get their views on the relative importance of COBOL optimization compared to other ADP management objectives and the appropriate management of COBOL optimization, (2) examine their documentation of benefits achieved, and (3) get an idea of the benefits that can be achieved with regular efforts of this type.
- Use of our earlier work and published results to augment our current experiments. (See app. III.)
- Discussion of the subject with organizations with a Government-wide interest in it, such as the GSA Compiler Testing Center.
- A limited experiment with a commercially available program, which we did to obtain our own data on automatic optimization--passing the machine code produced by the compiler through another program, called an optimizer, which eliminates unnecessary machine code and which reduces the size and increases the speed of the program.
- Examination of the current literature to get examples of reported benefits, to get examples of methods, and to assemble a bibliography for others' use. (See app. III.)

Our review was performed in accordance with GAO's current "Standards for Audit of Governmental Organizations, Programs, Activities, and Functions." We made this review because we had

indications--including an example reported in our software technology report 1--that optimization work can still yield worthwhile savings despite the fact that computing hardware is now cheaper and thus hardware costs do not have the overriding importance they once did.

We limited the review to COBOL because COBOL is the primary language used for the Government's high-volume business applications and because we believed that optimization would pay off most with a widely used language.

A list of the sites we visited, with summaries of what we found at each site, is included in appendix II. Officials at each site reviewed and commented on our summary of work done at their site soon after we finished our work.

---

1/"Wider Use of Better Computer Software Technology Can Improve Management Control and Reduce Costs," FGMSD-80-38, Apr. 29, 1980.

## CHAPTER 2

### SIGNIFICANT BENEFITS CAN BE ACHIEVED BY REDUCING MACHINE RESOURCES CONSUMED BY COBOL APPLICATIONS

Significant benefits have been achieved at some Federal installations by modifying COBOL applications to reduce the machine resources they consume. Many other installations, however, have done little or nothing in this area. The benefits of such work can be measured and verified. A systematic method and automated tools can increase the payoff. Work can be done to reduce the machine costs of COBOL applications on any brand of computer using COBOL, and new COBOL applications can be developed with deliberate attention to using machine resources more efficiently.

These recovered resources can result in significant savings and will sometimes defer the need to procure new computer equipment.

#### SOME FEDERAL INSTALLATIONS HAVE ACHIEVED SIGNIFICANT BENEFITS BY OPTIMIZING COBOL APPLICATIONS TO REDUCE MACHINE RESOURCE CONSUMPTION

The benefits from optimization may be divided into three categories. The first is the recovery of resources which can be applied to other uses. This work can recover CPU (central processing unit) time, 1/ file storage space, and the like, which can then be used for other applications. The Department of Housing and Urban Development (HUD), which has an ongoing effort, reported significant resource recovery. Figure 1 on the next page shows the results of work done by HUD on five application systems.

These five systems were the:

- Home Mortgage Distributive Shares System.
- Acquired Home Property Phase II System.
- Critical Path Processing System.
- Premium Liquidation and Control System.
- Small Homes System.

The first four were documented in HUD internal memos and in a June 1980 paper written by a member of the HUD Standards and

---

1/CPU time is the time the central processing unit spends executing instructions.

Figure 1

<u>System</u>	<u>Percent CPU time reduced</u>	<u>Per-year equivalent dollar value of CPU saved (note a)</u>	<u>Total cost to put optimized system into production</u>	<u>Net dollar equivalent of recovered CPU time savings first year</u>	<u>Approximate net dollar equivalent of recovered CPU time during minimum expected life</u>
A	82	\$37,000	\$5,500	\$31,500	\$142,500
B	30	4,400	2,400	2,000	15,200
C	19	9,000	900	8,100	35,100
D	45	45,000	1,200	43,800	178,800
E	9	7,000	9,000	-2,000	<u>19,000</u>
					<u>\$390,600</u>

Summary

For the five systems:

First-year dollar equivalent of recovered CPU time = \$83,400 (net)

Approximate dollar equivalent of recovered CPU time = \$390,600 (net)  
(based on 4-year minimum life)

Costs to put the five optimized systems into production = \$19,000

a/At HUD rate of \$350 per CPU hour. We did not verify this figure but believe it is reasonable.

Quality Control staff. 1/ The fifth, and most recent, was documented in an internal memo and discussed with us.

To estimate net savings from resource recovery, we augmented the HUD documentation with interviews of the HUD Standards and Quality Control staff. For example, we needed to estimate not only what the owners of the applications spent to put the optimized programs into production but what the HUD Standards and Quality Control staff spent as well. We were conservative in our calculations. For example, we used minimum instead of most probable expected life so that, if anything, the savings are understated.

The HUD representatives stated, and we agree, that not all systems will repay optimization efforts quickly and that not all expensive-to-operate systems are inefficient. As an example, the fifth system optimized did not repay the effort until the second year after the optimization.

During this review, our work produced two noteworthy improvements. First, with the cooperation of the Defense Mapping Agency, we modified a standard print routine, which is copied into many of the agency's COBOL programs, 2/ to improve its efficiency. We demonstrated a cost savings of \$1,357--using the improved print routine in one program--for the time that program would be run on the old computer. Later, after the agency had converted its programs to a new computer, we found that the improved print routine will be used in about 100 programs.

The other improvement came with the cooperation of the U.S. Army Management Systems Support Agency (USAMSSA) when we saved resources while correcting an operational problem. A program that manipulated three indexed disk files had about a 7-1/2 hour elapsed time to complete processing. During this long period, system breakdowns often caused the output to be lost, requiring re-runs. Users complained that they were not getting their output on time. By merely reorganizing the files, we cut in half both the elapsed time and CPU time consumed. Changes to the COBOL statements of the program (source code) cut them in half again. Thus, these changes--accomplished with a total of about 1 staff-day's work--cut CPU and elapsed times to one-fourth their original values. Also, the output is lost much less often and the users get their output on time much more often. The changes will also save about \$3,500 in CPU time the first year the improved version is operated. (See app. II.)

---

1/ "A Procedure to Review and Improve the Operation Efficiency of Production Systems," by Robert A. Grossman, C.D.P., Proceedings of the 19th Annual ACM/NBS Symposium, June 19, 1980.

2/ By means of the COBOL COPY verb.



An example from our software technology report 1/ further shows the benefits of resource recovery:

"All three programs we selected for this experiment were real applications programs \* \* \*.

"The projected first-year savings due to our experiment totaled about \$34,000 at the two installations. We spent about 4 staff weeks on the experiments. We also demonstrated that software tools developed at one installation can be used at another with high potential for cost-savings--avoiding duplicate development of tools--and with limited resources, and that the techniques which the tools aid can be applied at many installations."

The other two benefits that can be derived from tuning COBOL applications are a reduction in operational problems and improvements in software quality. In the first case, a new computer may be found to be filled unexpectedly by old applications which were converted without regard to the new system's capabilities. (We reported such a finding in our software conversion report. 2/) Work may then be needed to recover capacity. Also, on this project we corrected an operational problem at USAMSSA. (See app. II.) In the second case, tuning applications can simplify coding, eliminate unnecessary operations, and remove functional errors discovered during analysis or testing.

A published paper 3/ states:

"Application program optimization is an area often ignored by computer measurement personnel, who prefer to address the problem of tuning and optimizing the operating system software, finding it easier to measure and control. Yet this area offers substantial benefits to the computer facility as well as to the application. The advantage to the user is obvious, since an optimized program will cost less to run. However, on a well-tuned system, operating system software may be executing 30% of the time with the remainder of the CPU cycles in use by the application software. Thus, even the most finely-tuned operating

---

1/"Wider Use of Better Computer Software Technology Can Improve Management Control and Reduce Costs," FGMSD-80-38, Apr. 29, 1980, p. 26.

2/"Conversion: A Costly, Disruptive Process That Must Be Considered When Buying Computers," FGMSD-80-35, June 1980, pp. 12, 42-43, 48-49.

3/Budney, J., "COBOL Optimization Techniques," Proceedings, 17th Annual Technical Symposium ACM/NBS, Gaithersburg, Md., June 15, 1978, p. 221.

system may be degraded by inefficient applications. Any improvement in the efficiency of software which comprises 70 percent of a system's workload offers real savings to all."

Efforts to improve existing applications must select applications that are expensive or that are operational problems--so that the work will be worth doing. (See p. 14.)

SOME FEDERAL INSTALLATIONS HAVE DONE  
LITTLE OR NOTHING TO REDUCE THE  
MACHINE COSTS OF THEIR APPLICATIONS

Little done at some sites

Despite the potential for recovered capacity and operational improvement, we found that some installations have done little or nothing to examine the machine resource consumption of their COBOL applications. Several reasons were cited for this situation at sites we visited and in available literature. Among those reasons were:

- Cost of machine resources is not the users' chief concern and thus is not what the ADP shop is pressured about. At USAMSSA, the Director said he has an enormous backlog of user functional requests, no user concern for machine costs, and no labor to devote to optimization.
- Programmers do not perceive pressure from management, and are not self-motivated, about the costs of operating their programs. An NBS official said he felt that most COBOL programmers have the attitude that getting the program to work any way they can is satisfactory.
- Many COBOL programmers are not familiar with advanced features of and usage techniques for the COBOL language. The head of the GSA compiler testing center said he felt that most COBOL programmers are familiar with, at best, one-fourth of the features of the language and know little about how to use the language effectively and efficiently.

A published article 1/ summed up the situation:

"\* \* \* most programmers have little appreciation for the performance aspects of the programs they are writing. There are several reasons for this: (1) the programmers have typically never had any comprehensive training in the performance considerations of the programs; (2) the job reward system usually does not give an incentive for writing programs that give better performance; (3) management often has little or no idea as to what timeframe

---

1/Jalics, Paul, "Gaining An Awareness of the Performance of COBOL Programs," Computer Measurement Group IX, 1978, p. 61.

is reasonable for the execution of a given program; (4) the programmers are usually working with a higher-level programming language such as COBOL and are therefore almost completely shielded from the realities of the sequence of machine instructions that must be executed and the actual data types that must be manipulated; and finally (5) the language and compiler manuals are seldom found to be treasure houses of useful information and hints on how to get the best performance on the particular compiler."

#### Awareness and guidance found lacking at some sites

At the sites we visited, programmers' awareness and ADP managers' awareness of this area varied greatly. Some sites--such as HUD and the U.S. Army Computer Systems Command (USACSC)--made continual efforts to evaluate alternatives in new applications, to monitor costs of existing applications, to do cost reduction work where indicated, and to educate programmers in efficient techniques. Other sites had done little or nothing. Lack of knowledgeable staff and lack of authoritative guidance were cited, as well as general pressure from more important work. (See app. II.)

We discussed COBOL performance improvement with experts at GSA's Office of Software Development, FEDSIM, and at the Institute of Computer Sciences and Technology of the National Bureau of Standards. At the Office of Software Development and at FEDSIM, we were told that Government-wide guidance is needed to increase programmers' awareness of costs and their knowledge of techniques, and that both organizations would be willing to work with NBS on such guidance.

The NBS officials we spoke with indicated that many COBOL programmers are ignorant of techniques, get little formal training, and have little concern for machine costs. Concerning Government-wide guidance, the officials said that while NBS does not publish specific guidance on using COBOL for applications, general principles in some of their present publications apply to COBOL as well as to other languages. The officials also said it is not clear that guidance published by NBS would be read or followed any more than commercially published guidance. Our representative pointed out to the NBS representatives that NBS had already published specific guidance on using the FORTRAN language and if that was worthwhile, surely COBOL guidance would be also.

#### THE BENEFITS CAN BE MEASURED AND VERIFIED

Measurement and verification of benefits includes identifying applications that are now expensive to operate, recording labor and machine costs needed to prepare test data, planning and making changes to programs and files, making comparison tests of the old to the new version(s), and recording costs to change the documentation and turn the new version over to production. The benefits can be measured in terms of reduced machine resources and--for

comparison to the cost of making the improvements--expressed in dollar equivalents at local charge rates. Whether the machine resource reduction is truly a "cost savings" is more difficult to prove. For example, if the installation is committed to a procurement contract, its current payments for the computer will be largely fixed; running the programs faster will not reduce the money paid to the computer vendor.

However, even in the fixed procurement situation, expressing in terms of dollars the reduced resources needed will enable judgments of whether the tuning labor expended was justified. Also with fixed procurement, reducing the machine resources needed for the present applications will free up resources for other or later applications, thus allowing the computer to be used longer before its capacity is filled and a larger and/or more expensive replacement is needed.

#### A SYSTEMATIC APPROACH HELPS IMPROVEMENT EFFORTS

The advantages of using a systematic, well-documented method for this work include the following:

- Wasting time on applications that will not repay the effort can be avoided.
- Potential, worthwhile candidates can be quickly identified.
- Benefits achieved can be identified and quantified.
- Users will be more easily convinced that their improved applications will still get the right answers; for example, print the correct amount on the paycheck.
- The methods used can be recorded for later use by others.

The general procedure to reduce the machine resources consumption of an installation's inventory of COBOL applications is as follows.

- Identify large consumers. First, identify those applications that consume significant resources. Then, for the expensive applications, identify individual programs, files, or interactions which consume significant resources.
- Identify changes to programs, files, or interactions that might yield improvements, make those changes to working copies, and test them.
- If the changes do indeed reduce the consumption of machine resources, verify that the users' answers have not been changed.

- Turn the improved version over to production. After several production cycles, verify the savings.
- Throughout the process use appropriate tools to reduce the labor of analysis and testing.
- If the cost to make the improvement would not be paid for by saved machine resources, do not implement the change.

The general procedure is discussed in more detail in appendix I, the provisional checklist.

BENEFITS ARE POTENTIALLY WIDESPREAD AND AVAILABLE ON ANY BRAND OF COMPUTER THAT HAS COBOL

Many manufacturers of medium- and large-scale computers offer a COBOL language translator. Some minicomputers and microcomputers are also offered with COBOL. As we pointed out earlier, COBOL is a widely used Federal standard computer programming language.

Five of our case studies on the current review show the following percentages of production workload in COBOL:

<u>Site</u>	<u>Brand of computer</u>	<u>Estimated percentage of production in COBOL</u>
HUD	UNIVAC	97
NARDAC (note a)	UNIVAC	85
Corps of Engineers	Honeywell	90
Bureau of Mines	Burroughs	82
USAMSSA	IBM	60

a/Navy Regional Data Automation Command.

The fact that COBOL applications are significant in the production inventories of many Federal installations, and that long lives are often reported for COBOL applications, indicate the potential for significant savings. Even if an application's machine resource needs are reduced only slightly, the total savings will mount up if the program is run many times for several years.

Our own experiments, the sites we visited, and assorted literature showed us that the general systematic approach and many specific techniques can be applied to COBOL applications regardless of the brand of computer involved. Also, the GSA compiler test center director, representatives of NBS, and a FEDSIM representative agreed that this general applicability is true.

We performed optimization work at sites that were using IBM, Honeywell, UNIVAC, and Burroughs computers. We were able to demonstrate potential for significant reductions in computer resources by following the general techniques described above.

It is true that some optimization techniques involve the use of vendor-unique extensions to the standard COBOL language or other features unique to a particular vendor's hardware or operating system. Such techniques, however, should be used only for expensive applications where the eventual conversion cost increase due to their use is massively overbalanced by savings, and those savings should be well documented. Because of their impact on future conversions, such techniques should be rigidly excluded from applications that cost little to operate.

The head of the GSA compiler testing center and a FEDSIM representative gave us some thoughts on the appropriate use of vendor extensions and the kinds to use. (See app. II, part C.)

#### AUTOMATED TOOLS CAN HELP IMPROVEMENT EFFORTS

Automated tools are computer programs that can help efforts to reduce COBOL machine costs by reducing the labor cost needed to achieve the improvement. Automated tools for this purpose can be divided into two general categories--analysis aids and automatic optimizers.

Analysis aids include:

- Computer resource accounting aids which identify jobs by resource consumption. Some subdivide the information to the individual program level.
- System analyzers which can trace the flow of data and the consumption of resources through a group of programs making up an application and which can identify which programs in the group consume the most resources.
- Individual program analyzers which will quantify the behavior of individual programs by showing how many times parts of the program are caused to execute by a given data set, or how much time is spent in each part of the program, or both.

Analysis aids can provide useful information for identifying and tuning expensive applications. They may summarize information that would otherwise be laboriously gathered manually--such as a list of programs sorted in order of CPU time consumed. They may also collect information that is difficult or impossible to get manually--such as the percentage of a program's total CPU time that is spent in each part of the program. Better information from analysis aids reduces the labor cost of analysis and reduces false starts at improvement. Some of the information, such as

execution counts, may also locate logical errors. Analysis aids may be provided as compiler options or as separate processors.

Automatic optimizers include options built into compilers and separate processors, which act after the compiler. Automatic optimizers work directly on the object module 1/ produced by compilation and remove unneeded machine instructions and improve flow of control, thereby reducing the size of the object module and increasing the speed of the load module 2/ that will be link edited 3/ from it. Automatic optimization may be done as an additional phase of the compiler, or implemented as a separate processor which processes the object module after the compiler.

We have made four general observations that we believe apply to automated tools of either kind.

--Sources of tools: Use what is available. If at all possible, use tools or aids that already exist in the system of interest. If the desired aids do not exist on the computer of interest, search for tools that exist at other installations with the same type of computer and try to share a copy. If this fails, consider--with appropriate planning and testing--an available off-the-shelf software product aid. Tools should not be custom-built unless (1) demonstrated improvement potential is greater than the estimated cost to build the tool, (2) a plan for their use exists which includes a postinstallation audit to establish whether building the tool was worthwhile, and (3) serious research shows that there is no alternative.

--Plan for use: Do not acquire tools haphazardly with no plan for their use. USAMSSA bought COTUNE II--an analysis aid--several years before our visit and had used it very little.

--Availability of tools, experienced analysts, and information: Availability of these are much greater for some brands of computers than for others.

---

1/Object module is the output of the compiler, which translates the COBOL source statements written by the programmer to an intermediate machine-language object module.

2/Load module is the output of the linkage editor which links together the object code produced by the compiler and other programs and resolves references. It is ready for execution.

3/Link editing: The activity of the linkage editor in producing a load module. The linkage editor is a utility program which prepares an object module for execution by completing needed information.

--Whether commercially available tools are on the GSA schedule: GSA negotiates annual schedule contracts for commercially available software products, some of which are optimization tools. Federal installations should check the GSA schedule because a product can be quicker and easier to install if it is already on the GSA schedule.

The provisional checklist in appendix I contains more information on these points.

APPLICATIONS CAN BE DEVELOPED  
TO DELIBERATELY REQUIRE  
LESS MACHINE RESOURCES

Many of the same techniques used to reduce machine costs of applications after they have been running in production can be used when applications are first built. This would (1) yield the benefits earlier, (2) reduce the cost of getting the benefits because later extra testing would be avoided, and (3) improve the chances of discovering logic errors before the application is delivered to production. The last may provide users with more reliable as well as cheaper-to-operate software.

Our observations at sites, our discussions with GSA and FEDSIM, our earlier software technology 1/ and software maintenance reports, 2/ and NBS Special Publication 500-11 "Computer Software Management," all support the value of doing a thorough job early in the program development process.

We feel that teaching programmers to build new applications better offers a larger overall potential for improving COBOL applications than does the retrofitting of performance improvements to existing applications.

---

1/FGMSD-80-38, op. cit., p. 17.

2/AFMD-81-25, op. cit., pp. 16-21.



## CHAPTER 3

### EFFORTS TO REDUCE MACHINE RESOURCES

#### USED BY COBOL APPLICATIONS ARE NOT ALWAYS APPROPRIATE

#### AND MUST BE CAREFULLY CONSIDERED

The opportunities discussed in the previous chapter for reducing machine costs of COBOL applications are not always achievable. In some situations, it may not be practical or desirable to undertake such efforts; management must look at its own situation case by case. Some of the specific constraints identified during our review, and discussed in greater detail in this chapter, are as follows:

- Other objectives may have higher priorities.
- Efforts to reduce machine resource consumption may violate other objectives.
- Continual efforts to reduce machine resource consumption may not be cost effective.

#### OTHER ADP MANAGEMENT OBJECTIVES MAY HAVE HIGHER PRIORITY THAN ATTEMPTS TO REDUCE MACHINE RESOURCE CONSUMPTION

In applying its resources, ADP management must meet many other demands which can have higher priority than reducing applications' machine costs. These other demands might include:

- Satisfying user needs. User requests for new applications, or for changes to existing applications, may have higher priority and take up all the programmer/analyst labor that is available. The director of USAMSSA told us that his large backlog of user requests has prevented work to reduce machine costs of applications. However, even in this case, an automatic optimizer can help, because it requires no analyst labor after installation.
- Anticipating competitive procurement of new hardware or a new timesharing service. This may require software conversion to a different brand of hardware and can discourage improvement efforts on the current hardware. Some types of applications improvements indeed will not transfer to other hardware; others, however, will. At the Defense Mapping Agency, we found that improvements in the standard print routine that is copied into many of their COBOL programs would not only effect significant improvements on their present computer, but since the improvements were at the COBOL source level, would transfer to any make of computer that includes COBOL.

WORK DONE TO REDUCE MACHINE COSTS  
SHOULD NOT VIOLATE OTHER SOFTWARE  
MANAGEMENT OBJECTIVES

While reducing resource consumption is worthwhile, other considerations besides machine costs are also very important in software management, and these other considerations should not be violated in the name of saving resources. These considerations include:

--Making the software do what the user wants, and do it correctly. NBS Special Publication 500-11 states:

"Design and program for quality before performance.

"Before programmers make any effort to tune their modules for improved performance, all quality characteristics required should be realized. Necessary comments and program arrangements for clarity, maintainability, modifiability, generality, and all required functions should be completed and confirmed in reviews." 1/

--Constructing software with defenses against problem situations. Applications software should be constructed in such a manner that problem situations, such as attempts to input unacceptable data, will not cause results that are detrimental or astonishing to the user. Programs should include routine checks and recovery possibilities that are "forgiving" of common user and data errors. 2/ As an NBS official pointed out, (see app. II) such routine checks will add to the pure machine costs of running programs but are needed to reduce the chances of disastrous effects on user tasks. Attempts to reduce machine costs should not discard a reasonable level of defense in the application.

--Preserving maintainability of the software: A sizeable application often runs long enough in production status that user-requested changes--called maintenance changes--are made to the programs by persons other than their original authors. On our software maintenance questionnaire, 263 installations reported that their COBOL programs have an average life of 5.4 years. 3/ We found that significant

---

1/NBS Special Publication 500-11, "Computer Software Management: A Primer for Project Management and Quality Control," p. 31.

2/NBS 500-11, op. cit., p. 14.

3/"Federal Agencies' Maintenance of Computer Programs Is Expensive and Undermanaged," AFMD-81-25, Feb. 26, 1981, p. 44.

reductions in machine costs of COBOL applications can be made without affecting maintainability of the programs; indeed, some changes simplify programs and improve maintainability. Examples included our work with the print routine at the Defense Mapping Agency and at USAMSSA, and were confirmed by discussions with GSA.

--Following structured programming principles in constructing and maintaining programs. Structured programming makes a computer program easier to understand by arranging the procedural statements of a program 1/ hierarchically, using meaningful names for variables the program manipulates, using meaningful embedded comments in the program, and adhering to certain formatting and indentation conventions. Changes made to computer programs to reduce their machine costs should not, and need not, violate the principles of structured programming.

--Preserving adherence to the COBOL programming language standard. A Federal standard for COBOL is maintained by NBS so that programs will be more nearly the same on different brands of computers, with the objective that conversion costs and programmer training will be reduced. 2/ Vendors' COBOL compilers include features defined in the standard (most of the language) as well as additional "custom" features supplied by the vendor. Those custom features are commonly called extensions because they extend the capability of that vendor's COBOL beyond what can be done with purely standard COBOL. However, if and when COBOL programs containing extensions must be converted to another vendor's computer, those extensions must be taken out and standard language substituted. Our conversion report 3/ cited an instance where strict adherence to standard COBOL enabled a much cheaper conversion than other cases where nonstandard COBOL was present. Yet, in some cases, use of extensions or other vendor-unique features can significantly reduce the day-to-day operating costs of an application. Thus, a trade-off should be made: Will there really be enough immediate resource savings from using extensions to offset later conversion difficulty caused by those extensions?

---

1/The procedure division of a COBOL program.

2/Federal Information Processing Standards Publication 21-1, "COBOL."

3/"Conversion: A Costly Disruptive Process that Must Be Considered When Buying Computers," FGMSD-80-35, June 3, 1980, pp. 22-23.

In this study, we found that (1) significant reductions in the operating costs of COBOL applications can be made without deviating from the standard, (2) many COBOL programs do not cost enough to operate to justify any deviation from the COBOL standard, (3) certain COBOL language extensions and other vendor-unique properties will yield significant savings and should be used in expensive, long-lived applications, (4) some of those useful extensions can easily be removed, and (5) all use of such extensions should be documented, including resource reductions yielded, so that they can be easily removed later. These five points emerged from our own experiments, our discussions with NBS, FEDSIM, and GSA, and the literature we reviewed.

CONTINUAL EFFORTS TO REDUCE COBOL APPLICATIONS'  
MACHINE RESOURCE CONSUMPTION MAY NOT  
BE COST EFFECTIVE

To be worthwhile, this type of work must generate enough reduction of resources consumed to offset its own labor and machine costs. The people who can analyze existing applications and devise improvements are expensive to hire and should be supported with automated aids where feasible so that the labor cost of improvement efforts will not outweigh the improvements obtained. In this study, we observed that:

--Improvements seen will reach a point of diminishing returns. This is true at both the installation level and at the level of individual COBOL programs. At the installation level, the first work done to reduce resource consumption will (and should) select the worst applications. Spectacular improvements with a good return for resources expended may be seen on these first, worst applications. However, later work on other applications that are not as bad may not show nearly as good a return. This phenomenon of diminishing returns was reported by HUD and the Bureau of Mines, Denver (see app. II), and we believe that it is closely tied to the "80/20 syndrome" observed in resource consumption. (See p. 23.)

At the level of individual programs, a similar phenomenon is seen. The first, most obvious changes to a program, its files, or its interactions with its environment may, and often do, yield a far greater resource reduction than other, less obvious, more difficult-to-make changes. Thus, those responsible for efforts of this type should "know when to stop."

--In smaller installations, resource reduction efforts may be part-time or periodic. A small installation may not be able to justify continuous full-time staff devoted to this work. This can be particularly true after an initial concentrated effort to reduce the consumption of the worst

applications. The Bureau of Mines site in Denver had experienced just this sort of thing. A highly skilled employee did such work full-time just before he retired and reported to management that the worthwhile work was done. The director at that site said that another resource reduction effort would probably be justified in another 2 to 3 years due to new applications and the degradation of current ones. The branch chief said that changes would eventually accumulate to the point that another optimization exercise would be needed.

- Large installations, however, can benefit from continuous effort. Large sites have enough potential benefit to justify continuous effort of this type, especially if they develop applications that will be run at many sites. Even a small percentage reduction in resource consumption becomes worthwhile when it is multiplied by 10 or more sites. The U.S. Army Computer Systems Command develops software for many Army sites and has a continuous resource reduction effort. (See app. II.)
- Sporadic operational problems may be corrected in ways that also reduce resource consumption. Systems people at USAMSSA asked us to look at a problem for which they did not have time. As reported on page 15, we resolved this problem for them.

If an application costs so little to operate that improvement of the application will not offset the cost of making the improvement, then nothing should be done. In this regard, we found that many applications do not cost much to operate. A very common phenomenon is that a fraction of the total number of applications consumes a disproportionately large amount of the machine resources. This general phenomenon is sometimes referred to as the 80/20 syndrome, that is, 80 percent of the resources are consumed by 20 percent of the applications. (This rule, inverted, is also known as the 20/80 rule. The proportions vary, but the general phenomenon is common. A FEDSIM representative said that (1) it is typical (see app. II, part C), (2) published papers report it, 1/ and (3) our own analysis at the Defense Mapping Agency and USAMSSA showed about 70/15 and 77/10 respectively. (See app. II, part B.)

This means that in many installations, half or more of the COBOL programs may not consume enough resources to justify any expenditure of analyst labor to change and retest them to reduce their resource consumption. (However, such programs can be helped somewhat by automatic optimization which requires no handwork.)

---

1/Grossman, "A Procedure to Review and Improve the Operational Efficiency of Production Systems," Nineteenth Annual ACM/NBS Technical Symposium, p. 222.

An official of the House Information Systems said that their programmers normally use automatic optimization on applications before releasing them to production.

Other circumstances may preclude optimizing applications. For example, if applications software is soon to be redesigned, replaced, or upgraded, efforts to optimize it may be wasted motion. Also, anticipated conversions to new hardware environments will discourage tuning on the present hardware. Some types of changes to reduce resource consumption will not transfer to a different brand of hardware. However, other types of changes will be better on any brand of hardware, as we demonstrated at the Defense Mapping Agency.

## CONCLUSIONS, RECOMMENDATIONS, AND AGENCY COMMENTS

### Conclusions

Significant computer resources can be recovered for other purposes when work to reduce consumption by COBOL applications is properly done. The potential aggregate benefit to the Federal Government is quite large because of the widespread use of COBOL, the long lives of many COBOL programs, and the fact that some installations have done little to reduce resource consumption. More needs to be done to raise ADP managers' and users' concern with the cost of applications and to raise programmers' efficiency and effectiveness in building and maintaining COBOL applications. Agencies with Government-wide ADP responsibilities should publish guidance on reducing machine resources consumed by COBOL applications. We believe that COBOL applications deserve separate treatment because of the extensive Federal use of COBOL.

The general methods of managing and doing work to reduce COBOL applications' consumption of machine resources can be used by any agency on any brand of computer that implements COBOL. This work should be done only when the recovered resources will significantly repay the cost of doing the work. Many of the techniques used from one brand of computer to another are similar and, once learned, relatively easy to apply. Efficient machine resource usage can be built into new COBOL applications or retrofitted to old ones.

Efforts to reduce machine resources consumed by COBOL applications need not violate other software management objectives such as maintainability, ease of conversion, and adherence to language standards.

### Recommendations

We recommend that the Secretary of Commerce direct the National Bureau of Standards to publish guidance on the effective and efficient use of COBOL for applications. We believe that the guidance should include examples taken from real-life applications. We also believe that a possible starting point would be to use a

table of contents similar to that of the already published "Using ANSI FORTRAN," <sup>1/</sup> and our provisional checklist (app. I). We also believe that GSA's Office of Software Development and FEDSIM could work with NBS in constructing such guidance.

Heads of Federal agencies should require periodic review of the machine resource consumption of COBOL applications at their installations, and, where feasible, require action to reduce the consumption of the expensive applications.

#### Agency comments

We asked for comments from the National Bureau of Standards of the Department of Commerce, the General Services Administration, and the Federal Computer Performance Evaluation and Simulation Center. All three furnished comments which are included verbatim in this final report as appendix IV. Officials at the sites we visited were allowed to review and discuss our summaries soon after we finished our work. We did not ask for comments from the parent agencies of our experimental sites because our recommendations were not addressed to them.

GSA said that its Office of Software Development would follow our recommendation to work with NBS in publishing guidance on the effective and efficient use of COBOL and would also continue its program of providing further guidance and assistance to agencies in improving their software. GSA also suggested further emphasis on testing of the improved software and pointed out a document of its own that addresses testing. We made minor changes to the report to recognize GSA's concerns.

The National Bureau of Standards officials did not concur with our recommendation that they publish guidance on the effective and efficient use of COBOL for applications because they judge language standards, total system performance, and life cycle management to be more important. NBS officials said they were able to publish a FORTRAN handbook only because they found an outside expert who was willing to contribute, and concluded that the language-by-language approach to improving software was not feasible. They also suggested wording changes to our account of our discussion with members of their staff and suggested that we add some NBS publications to our reference list.

We believe that specific guidance on COBOL is worthwhile because of the extensive Federal use of COBOL and that the General Services Administration may be able to publish the guidance if NBS does not. While we agree with NBS that language standards, total system performance and life cycle management are important, we believe that working-level Federal programmer/analysts need practical guidance on their primary language. We also recognize the resource

---

<sup>1/</sup>NBS Handbook 131.

limitations; however, we believe that assistance and existing documents are available to reduce the resources needed to publish such guidance.

Indeed, one of NBS' own recent publications says:

"Use of assembly language programming has been decreasing and the use of COBOL has been increasing. It was estimated that over 50 percent of Federal installations were using COBOL as their principal programming language in 1979. This number was predicted to increase to over 60 percent by 1985." 1/

We believe that a programming language that is expected to increase in use until it becomes the principal language of over 60 percent of Federal installations deserves a specific guidance document and the commitment of resources to produce it.

NBS officials suggested some rewording, but we made no changes based on their suggestions. We believe that we have presented the material accurately, and we feel that the points they suggested we delete are relevant to this topic. We have, however, added their publications to the reference list. They appear on pages 61 and 62.

FEDSIM had no comment on the substance of the draft report. They requested that we clarify that the person we spoke to was not expressing a FEDSIM agency position. To clarify, we have changed the wording on page 57 from "a FEDSIM representative" to "a member of the FEDSIM technical staff."

---

1/NBS Special Publication 500-79, "An Assessment and Forecast of ADP in the Federal Government," National Bureau of Standards, Washington, D.C., Aug. 1981, p. ix.



PROVISIONAL CHECKLIST FOR REDUCING MACHINE  
RESOURCES CONSUMED BY COBOL APPLICATIONS

For Guidance of ADP Managers, Programmers, and  
Analysts Responsible For COBOL Improvement Efforts

	<u>Page</u>
Introduction	28
Appropriate level of effort	28
The general procedure	28
Costs and benefits of performance improvement	31
Selection of existing applications to tune	33
Construction of new applications	33
Specific techniques and considerations	34
Tools considerations	42
Interaction with the users of applications	44

## INTRODUCTION

This checklist, which we prepared during our review, lists matters that we feel agency ADP management and programmer/analysts should consider to reduce the machine resources consumed by their COBOL applications. A few specific technical considerations are included because we feel they are very useful.

While this checklist is only an interim document pending publication of NBS guidance, we feel it will be useful to persons involved with COBOL applications. The levels of effort and emphasis devoted to specific items mentioned will vary with the type and size of specific applications and installations. Programmers and analysts should have no difficulty in grasping these matters and formulating an appropriate approach to their use.

## APPROPRIATE LEVEL OF EFFORT

Recognizing that this type of work is easy to delay because of the many "higher priority" demands for skilled labor, we suggest that a small segment of time be set aside every year during which some effort, albeit small, is dedicated to measuring and reducing the machine resources consumed by COBOL applications.

Periodically measuring and recording the resources consumption is, we believe, the minimum level of effort. After establishment, it can be done with little effort and will enable trends in consumption to be identified. Such measurement will also provide input for capacity management planning and for equipment decisions.

Larger installations can and should commit continuous resources; for smaller ones, yearly measurement and an "overhaul" of the most expensive applications every 2 to 3 years are more appropriate. In all cases, the labor and computer time spent on tuning should be recorded and compared to the resources saved, "valued" at local charge rates. If the work does not make a "profit," it should be reduced.

## THE GENERAL PROCEDURE

For a given inventory of applications at an installation, the general steps would be:

- Examine the application inventory to see if it includes much COBOL.
- Examine the log data to determine which applications cost the most to operate, including CPU and input/output (I/O) times for executing the program(s) and storage costs for the files. A smaller application that runs frequently may use more resources than a large one that runs once a year, so the variables to be considered include frequency of execution, resources consumed per execution (CPU and I/O

time), and constant consumption (file storage). For purposes of decisionmaking and comparison to labor costs, annual dollar cost of machine resources can be calculated at local charge rates.

- Identify those expensive applications that appear to have some potential for optimization. This includes impressions from persons familiar with applications--for example, someone may know that a certain expensive application has already been tuned and that it is not likely to reward optimization effort.
- If some applications with improvement potential are identified, plan changes that seem to offer improvements, then plan instrumentation and comparison tests and execute those tests. Types of changes to be considered include changes to the COBOL source code itself, reorganization of the files processed, and changes to the COBOL programs' interactions with external entities--such as the SORT/MERGE package 1--which the programs must interact with. Instrumentation possibilities include timings and execution count data on a program's execution, information on file activity in a typical run, and information on time the application spends in entities outside its COBOL programs. Labor and hardware costs of this analysis and testing should be recorded.
- If a new version is better, demonstrate by tests and comparisons that the users' answers are unchanged from those of the previous version(s) of the program(s); subject the new version to normal acceptance testing and documentation (as done with new development). Record these costs.
- Place the new version into production and run at least one production cycle in parallel. The extra cost of doing this parallel run of old and new versions should be charged as one of the costs of the optimization work.
- After several production cycles, verify that the new version is indeed better than the old one.
- Besides programs and their files, consider the efficiency of external items widely used in, by, or with the COBOL applications. Examples include the SORT/MERGE package used, standard routines which are replicated in many programs by use of the COBOL COPY verb, and subroutines which many COBOL programs CALL. (Our experiment with the standard

---

1/The SORT/MERGE package is a utility program which sorts records into ascending or descending order (as specified) on specified keys--for example, ascending order on Social Security number. The COBOL verb SORT communicates with this package.

print routine at USAMSSA illustrated a COPY text improvement, and others reported improvements with better use of SORT/MERGE. 1/ This illustrates the potential of such consideration of external items.)

At the level of individual programs, a published paper 2/ explained the general methodology as:

"Methodology

A program chosen as a candidate for optimization should be debugged and thoroughly tested. The input data for use during the optimization process should be live data representative of a typical run; its blocking and organization should not be altered except in the course of the optimization itself. This is critical, since the inspection of the coding in itself tells little about the run characteristics; it is the interaction of the program and its data which is the subject for measurement, analysis, and improvement. 2/

"The general methodology for optimizing a COBOL program is as follows:

1. Determine where the program is spending its time by applying an appropriate measurement tool. Direct optimization efforts at those areas which are most critical in order to achieve the maximum gain for the minimum effort.
2. Apply efficient COBOL coding techniques to the program making source level changes to minimize format conversion, loop housekeeping, and unnecessary data manipulation in the generated code.
3. Apply efficient data management techniques to the program, if it is I/O bound, to reduce elapsed time. This may involve changing access methods, block-sizes, job control language, and source level statements.

When the process first begins, each coding and data management technique applied will 'buy' an obvious improvement in efficiency. As optimization"

---

1/Grossman, "A Procedure to Review and Improve the Operational Efficiency of Production Systems," Nineteenth Annual ACM/NBS Technical Symposium, Gaithersburg, Md., June 19, 1980, p. 224.

2/Budney, op. cit., p. 222.

"proceeds, however, the gains received for the effort will decrease. At some point, each program change will result in little, if any, improvement; at this point, the program is, essentially, optimized.

4. Use an object-level optimizer to produce a more efficient--and smaller--load module. 1/ 2/
5. Execute the optimized program on appropriate test data and compare the outputs with those of the unoptimized program. This can be accomplished using a file/data comparison utility, allowing the computer to do the actual comparison of the data and report differences between the outputs of the old and new versions of the program. This will help verify that the optimized program gives results identical to those of the unoptimized program and ensure that the integrity of the program has not been lost."

#### COSTS AND BENEFITS OF PERFORMANCE IMPROVEMENT

Performance improvement is done to reduce the machine resources consumed by applications or to correct operational problems. Sporadic operational problems may simply demand correction; however, continuous improvement work to save machine resources must "make a profit" on the resources expended to get the improvement. This is why the appropriate course for smaller installations may be to have an "applications overhaul" every 2 to 3 years with little or no continuous effort in between.

The costs of performance improvement typically include both machine time and analyst time for the following purposes:

- Identifying expensive applications.
- Discussing them with their "owners," including cost history, processing cycles, special uses, or anomalies encountered.
- Preparing test materials, including working copies of programs, test data, documentation, and test tools.

---

1/Budney, op. cit., pp. 226-227.

2/The load module is the output of the linkage editor which links together the object code produced by the compiler and other routines and resolves external references. The load module is ready for execution.

- Planning test runs.
- Analyzing test run data.
- Running additional tests if needed.
- Convincing users that the modified application (1) is better in the machine and (2) processes their data in the same way as the version they are now using.
- Documenting changes made and predicated savings.
- Running in parallel operation with the current version for at least one production processing cycle.
- Turning over to production.
- Following up after 6 months to a year to verify savings.

The benefits of performance improvement can include:

- Savings in machine resources which can be used for other tasks.
- Deferred procurement of hardware.
- Better user service, such as quicker turnaround.
- More reliable software due to discovery and removal of old errors or omissions.
- Training development and maintenance applications programmers because of their interactions with skilled performance improvement analysts.

When evaluating performance improvement work:

- "Value" at local charge rates machine resources recovered and the machine resources used for the improvement.
- Value analyst labor at salary plus overhead.
- Consider how long the improved application will last before (1) functional maintenance changes cause it to be so "different" that the calculated savings are no longer valid or (2) it is replaced or discarded.
- Use a one-year maximum payback period as the general criterion for judging whether the improvement work is worthwhile.
- Keep good records of benefits realized and methods used.

### SELECTION OF EXISTING APPLICATIONS TO TUNE

Existing applications must be judged as "worthwhile" to be selected for tuning. Worthwhile may mean correcting a user's operational problem, such as too slow turnaround, or it may mean recovering significant machine resources.

Applications that cause user problems will be easy to select because the users will complain about them. A regular, planned effort to recover machine resources, however, should focus on high cost applications ("resource hogs") because it is in those applications that such efforts can pay for themselves.

A useful aid is to produce lists of the applications sorted in several different orders, including:

- Descending order of CPU time.
- Descending order of amount of file storage media used in a year's processing.
- Descending order of maximum requirement for highspeed memory (to run the largest program in the application).

Once the expensive applications are identified, then, within the applications, the expensive programs can be identified. Very often, one or two programs and their files--in an application that may consist of dozens of programs and files--may consume most of the resources that the entire application consumes. The general 20/80 rule applies: to applications within a group of applications, to programs and files within a group of programs and files, and to paragraphs within a single program. The 20/80 rule is that a few consumers (for example, 20%) very often consume a disproportionately large share of the resources (for example, 80%).

Thus, effective resource recovery work requires successively narrowing down the attention to applications that are likely to repay improvement work and ignoring the other, cheaper-to-run applications.

### CONSTRUCTION OF NEW APPLICATIONS

Several principles apply in the construction of new applications. Correctness is paramount: Users will not take kindly to an application that runs faster on the machine but deletes, for example, the accounts receivable file from a customer data base. More generally, other software quality characteristics, including correctness, maintainability, error/exception handling, and portability must be considered and successfully resolved before machine efficiency.

Many of the techniques and considerations that can be retrofitted to existing applications to recover resources can also be used in constructing new applications. Some of the more efficient alternatives, once learned, are as easy to use as less efficient ones, especially if their use is planned before any source code is actually written.

Other techniques must await some actual production history before applications can be fine-tuned with them. An example is the question of how often an indexed file must be reloaded to maintain acceptable response times. Another example is that of verifying the data profile type of performance improvement; for example, rearranging a searched table according to hit probability.

Vendor-unique and/or device-dependent considerations may be unavoidable; for example, (1) file block sizes which are chosen to closely fit the track size of a given disk device, and (2) use of vendor-unique COBOL source language to provide in-core indexes for quicker retrieval from indexed files.

#### SPECIFIC TECHNIQUES AND CONSIDERATIONS

This section discusses some specific techniques and considerations that often arise when COBOL optimization is done.

##### Program modification (source code)

###### Structured programming

This method results in better planned, better organized programs which are more maintainable (see above) and which have a better chance of avoiding redundant or erroneous code.

When structured programming was introduced, it was said that structured programs might be less efficient but that reduced maintenance labor costs would more than make up for possible increases in machine costs. However, further use and investigation of structured programming have led us to believe that not only does structured programming not conflict with machine efficiency, but that it probably yields better machine efficiency for realistically sized, long-lived applications. The head of the GSA compiler testing center said that with some COBOL programs tested there, structured programs were more efficient than unstructured ones doing the same tasks. And an IBM paper 1/ said:

---

1/Capers Jones, "Optimizing Program Quality and Programmer Productivity: The Improved Programming Technologies," SHARE 50, pp. 15-16.



"When I first started exploring the new programming technologies, my expectation was that a topdown, structured program written in a high-level language would be considerably larger and would execute more slowly than an unstructured program written in BAL. [1/] To my surprise, when I looked at the results of re-programming some 'old style' programs into topdown, structured form some of them (but not all) actually were smaller and executed more rapidly with the new technologies than they did before.

\* \* \* \* \*

"When I looked into the 'complexity' issue to find out why many old-style programs were so complicated, I discovered that much of the complexity seemed to be unnecessary, and was a result of bottom-up, unstructured methodologies rather than a result of true business or technical needs. \* \* \* The overall situation regarding the impact of the improved programming technologies on performance and execution speed is ambiguous today. Some programs seem to improve or speed up their performance when redone with the new methods, while others may grow larger and slower. The important aspect of the situation, however, is that the automatic assumption that top-down structured programs written in high-level languages will be larger and slower than unstructured BAL equivalents seems not to be generally true." (Underscoring provided.)

Thus, machine efficiency should not be an excuse for destructuring a structured program. Also, of course, the additional maintenance labor caused by destructuring would very likely far outweigh possible gains in machine efficiency.

#### Choice of algorithms

A better algorithm in standard COBOL will be more efficient than a worse algorithm that uses vendor-unique features in an attempt at speed. An example is table search algorithms: COBOL provides the SEARCH--WHEN construct for linear searches, and the SEARCH ALL--WHEN construct for binary searches. Yet programs frequently include clumsy, handcrafted searches (constructed of iterative loops and the like) that could be better handled by the built-in constructs. Not only are the built-in constructs often more efficient than the handcrafted searches, but the built-in constructs are also much better for the maintenance programmers--they always work the same way because the COBOL standard requires them to do so and they are shorter.

---

1/Basic assembly language--IBM's term for their assembly language.

Specific techniques should be followed for table search improvement, as follows.

- See if the search needs to be done at all, or as often as it is now being done.
- If all cells of the table have about equal probability of being the target of a given search and the table has over 30 cells, use a binary search: the SEARCH ALL--WHEN construct.
- If a few cells of the table have very high hit probabilities, compared to the other cells, either handle them specifically with IF tests before entering the general SEARCH construct or put them first in the table and use a linear search (the SEARCH--WHEN construct). Examination of execution counts from test runs will often identify which cases are "hit" most often if the test data is representative of production data.
- If the table is large and well ordered (for example, sorted) a partitioned scan can be used. This technique first isolates the location of the search argument to a partition of the table, then searches within that partition to find the argument.
- Generally, avoid handcrafted searching techniques and, if possible, replace them in existing programs. In many situations they are inferior to the standard SEARCH--WHEN and SEARCH ALL--WHEN constructs for two reasons: (1) maintenance programmers have difficulty with them because they are non-standard and must be deciphered and (2) they often are less efficient than the standard constructs.

#### Use of vendor-unique source code.

Generally avoid using nonstandard source code (extensions). Especially avoid useless items such as nonstandard spellings of reserved words. Unless clearly justified by savings, avoid using packed or other nonstandard data representations. Extensions that are justified and very useful include those that allow file blocking information to be supplied by the operating system so that the same program can process files with different blocking factors without recompilation.

#### Input/Output

##### Selection of file organization.

Standard COBOL (1974) now supports three file organizations: SEQUENTIAL, RELATIVE, and INDEXED. The last two were included to support efficient use of direct-access devices such as disks. The older 1968 standard supported only SEQUENTIAL organization, forcing

applications to use vendor-unique COBOL Input/Output (I/O) features in order to effectively use direct access devices.

Modifying the file organization of an existing application requires great care: Several programs may use the same file(s) and, to avoid redundant files (for example, two copies of the same information organized differently), reorganization of the file for better performance with one program may require that other programs be modified to handle the new file organization. Also, the storage devices must be loaded with the file(s) and tested. The complications may preclude reorganizing the files of an existing application.

For new applications the choice of file organization should consider hit ratio(s) anticipated, types of devices, (both available and anticipated), and the anticipated growth of the file(s). We believe that many COBOL programmers are ill-informed about the nonsequential file organizations and thus use SEQUENTIAL organization for situations to which it is less well suited than a nonsequential organization would be. They use SEQUENTIAL because they know it and can get something working more quickly with it.

Vendors' programmer guides often include detailed advice on the organization and processing of files, and should be consulted, preferably before the application is coded.

#### Blocking and buffers.

The practice of blocking records consists of having more than one logical record 1/ in each physical record. 2/ With larger blocking factors, 3/ fewer physical READs of the storage device are needed. A related matter is buffers. Buffers are holding areas in main memory into which information is transmitted from the file device in readiness for processing by the application program. Significant reductions in I/O time charged to run the program and in space required on file devices have been achieved with reblocking and more buffers in some cases. And, if the COBOL programs were originally written to be device independent, this improvement can be accomplished without recompiling the COBOL source. Vendors' programmer guides contain guidance and such reblocking of files is often supported by vendor utility programs.

---

1/A logical record is the record seen by the COBOL program.

2/A physical record is the information transferred by one physical READ of a device (for example, one motion of a tape). It may equal one or more logical records.

3/Blocking factor is the number of logical records per physical record.

Reloading indexed files.

Significant improvements in reduced elapsed time can be achieved with reorganizing indexed files on their storage devices. An example is our work on the program at USAMSSA. Such reorganization is needed after a file has been used to reload overflow records into prime data areas and, perhaps, to recover record areas that have been logically deleted, but still occupy physical space. Also, for large indexed files, the physical location of the index, prime, and overflow areas can have significant effects on performance. 1/

Use of vendor-unique COBOL features for I/O.

This should be avoided if possible, especially in the construction of new applications. When used, vendor-unique COBOL source language features should be well documented so that they can be found and removed if a later conversion to non-plugin-compatible hardware is necessary. Also, choices, such as block sizes, which relate to capacities of specific storage devices should be well documented because the files may eventually be migrated to other devices whose different properties may have performance implications.

Some vendors offer COBOL source language extensions which provide very helpful I/O improvements. Examples include language that causes file indexes to be loaded into main memory for faster searches and language that causes the data management routines to provide information for judging the worth of reorganizing an indexed file.

External entities

External entities to programs may be used to enforce standardization or to reduce coding labor for tasks needed by many programs. External entities include subroutines, COPY text, and the SORT/MERGE utility. Generally, if an external entity is used by many programs, then relatively small improvements in the entity will be significant because they will be multiplied by the number of programs. Also, if a COBOL program's main purpose is to "feed" an external entity (for example, a COBOL program which communicates with the SORT/MERGE utility via the COBOL source SORT verb) then most of the CPU time charged to executing the COBOL program will actually be spent in the external entity.

---

1/Newer types of indexed file organization can supply automatic reloading.

### Subroutines.

These are separately compiled subprograms which a main program invokes with the CALL verb. They offer the following advantages:

- Modular programming. A subroutine can only access data items that are made available to it through linkage; the rest are protected. This can greatly aid debugging.
- Division of labor. The use of subroutines allows large programs to be divided among several programmers. All the writer of a subroutine need know about the rest of the code is what data items are to be communicated to and from it.
- Security. Programmers can use a subroutine (by having their programs CALL it) without knowing the logic of the subroutine; all they need do is communicate the appropriate data items for the task to the subroutine.

However, a CALL to a subroutine is often slower than a PERFORM of an internal paragraph. If a subroutine is CALLED many times, a CPU saving may be achieved by turning it into a PERFORMED paragraph in its former CALLer.

### COPY text.

The COBOL language includes a COPY verb that causes actual COBOL source language to be inserted before the program is compiled. Text COPYed may be either procedural or nonprocedural. If a COPY text is used by many programs, it should be reviewed for performance implications. We were able to demonstrate significant improvements at the Defense Mapping Agency by changing a standard print routine used by many programs.

### SORT/MERGE.

The SORT/MERGE utility is commonly invoked by the COBOL SORT verb. It can consume a major part of the CPU time charged to the program that invokes it. Ways to improve sorting include:

- See that only the records that need to be sorted are passed to the SORT/MERGE. For example, select them in an INPUT PROCEDURE in the COBOL program.
- Ensure that the SORT/MERGE has adequate work space areas.
- Procure a more efficient SORT/MERGE utility--for some computer vendors, independent software houses supply more efficient SORT/MERGE utilities to replace the hardware vendor's utility.

Some vendors provide COBOL source language extensions for communicating with the SORT/MERGE. Their use should be well documented. Vendors' COBOL programmer guides often include advice on efficient sorting.

#### Test data for performance improvement

Test files are important in performance improvement work. The test files must correctly represent the production files in terms of file organization, location on devices, and frequency distribution of data. For example, if the production file is organized sequentially, its test counterpart should also be sequential; if the production file is spread over three devices, its test counterpart should be also; if the production file, for example, contains 30 percent employees with 4 dependents and 20 percent employees with 5 dependents, its test counterpart should have the same proportions. If the test files do not accurately represent the production files, then the "improvements" seen during testing of changes may not be reflected in production.

Sometimes, working copies of production files can be used for performance improvement testing. Often, however, testing with the complete production files would cost too much, and they must be abstracted to prepare test files. Preparing a test file by copying every third or every tenth record from the production file(s) is often sufficient. Frequency distribution phenomena, however, should be verified. Test files can often be prepared with utilities.

#### Automatic optimization

The great attraction of automatic optimization is that it takes no analyst labor--a processor does the work. Automatic optimization reprocesses the object module produced by the compiler to eliminate unnecessary instructions. Automatic optimization may be provided with compiler options or with separate processors. Separate-processor automatic optimizers have been particularly successful on IBM and plug-compatible mainframes. House Information Systems told us that the one they use had paid for itself very quickly and that they ran all their production COBOL programs through it.

#### Paging environments

Some brands of computers provide memory paging in which programs are divided into "pages" and those pages not currently executing are not in the main memory--they are brought in when it is "their turn."

Some COBOL programs spend a lot of time transferring pages in and out of main memory to the detriment of accomplishing useful work. This phenomenon, called "thrashing," can be reduced by rearranging the programs' source code. Vendors' programmer guides include advice on this subject.

Paging can be controlled by (1) rearranging the program source code to take advantage of the known behavior of an automatic paging mechanism or (2) actively controlling the paging mechanism by changing page size or overriding the page assignments and priorities of parts of the program. Depending on the hardware vendor, operating system, and compiler, this may be done with control language, linkage editor, and/or COBOL source extensions.

The production effect of paging optimization can depend upon what the mix is--what other programs are contending for resources. If no other programs contend for resources (for example, on the third shift) a theoretically better paging arrangement may yield little practical benefit.

We believe that deliberate tuning of paging should be avoided in many cases because it can make the application's performance dependent on the operating system, the compiler, or on the mix in which it is operated, with adverse impact on later conversions.

#### Executions of more than one program in succession

Production job streams may cause several programs to be executed in succession without human intervention. The first program to execute may output files which are the input to the programs executed after it. If, in this case, the first program does not execute correctly, succeeding programs' executions will be useless.

A way to avoid useless execution of the second, third, etc., programs in the job stream is to have the first program communicate a "go ahead" signal to its successors. This may be achieved either by having the first program write a special signal record which is read by the second program before it starts processing, or by using vendor-unique source language to communicate with the operating system's control language to allow, or not allow, the successors to be executed. Use of vendor-unique language for this purpose can be cheaper and more convenient than a special signal record file, but it must be carefully documented. It does "lock-in" the application to the specific vendor, so it must be changed when the vendor changes.

Another cost reduction can be achieved through "run stream optimization" in which a run stream (job stream) that executes several small COBOL programs with the passing of many small, intermediate files, may be improved by combining the COBOL into fewer, larger programs and eliminating some of the intermediate files. This method can yield handsome improvements but requires redesign of programs, some recoding, and much testing--it is not a "quickest band-aid fix" as some of the other methods are.

TOOLS CONSIDERATIONS

Automated tools used can be divided into three general categories--the log, analysis aids, and automatic optimizers.

Vendors normally provide a file which captures log data on what was run on the machines. The log data (1) provides a history of what was executed and (2) can be used to identify very expensive single jobs, jobs with very long elapsed times, and programs which are executed very frequently when its records can be sorted by CPU time consumed, by elapsed time, and/or by program/job name. Such manipulations will provide a "most wanted list" of applications--those that consume the most resources. Software packages or relatively straightforward, locally written extraction programs can greatly aid analysis of the data in the log file.

Analysis aids can provide useful information for identifying and tuning expensive applications. They may summarize information that could more laboriously be gathered manually--such as a list of programs sorted in order of CPU time consumed. They may also collect information that is difficult or impossible to get manually--such as the percentage of a program's total CPU time that is spent in each part of the program. Better information from analysis aids reduces the labor cost of analysis and reduces false starts at improvement. Some of the information, such as execution counts, may also locate logical errors. Analysis aids may be provided as compiler options or as separate processors.

Analysis aids include:

- Computer resource accounting aids which identify jobs by resource consumption. Some subdivide the information to the individual program level.
- System analyzers which can (1) trace the flow of data and the consumption of resources through a group of programs making up an application and (2) identify which programs in the group consume the most resources.
- Individual program analyzers which will quantify the behavior of individual programs by showing how many times parts of the program are caused to execute by a given data set, or how much time is spent in each part of the program, or both.
- Testing tools which can, for example, automate the comparison of outputs produced by the old and new versions of programs that have been optimized, to verify that the optimized one gives the same user results. 1/

---

1/Testing is discussed in "Software Improvement--A Needed Process in the Federal Government," Report OSD-81-02, Office of Software Development, ADTS, GSA, June 3, 1981.



Automatic optimization may be done as an additional phase of the compiler, or implemented as a separate processor. Working directly on the object module 1/ produced by compilation, automatic optimizers remove unneeded machine instructions and improve flow of control, thereby reducing the size and increasing the speed of the load module 2/ that will be link edited 3/ from that object module.

General observations that we believe apply to automated tools of either kind include:

- Sources of tools. Use what is available. If at all possible, use tools or aids that already exist in the system of interest. If the desired aids do not exist on the computer on the computer of interest, search for tools that exist at other installations with the same type of computer and try to share a copy. A suitable tool may also be available from the Federal Software Exchange operated by GSA, and they should be consulted. If this fails, consider--with appropriate planning and testing--an available off-the-shelf software product. Software tools should be custom-built only when (1) demonstrated improvement potential is greater than the estimated cost to build the tool, (2) a plan for their use exists which includes a postinstallation audit to establish whether building the tool was worthwhile, (3) serious search shows that there is no alternative, and (4) their eventual use at other installations is possible.
- Plan for use. Do not acquire tools haphazardly with no plan for their use.
- Availability of tools, experienced analysts, and information. These are much more readily available for some brands of computers than for others.
- Whether or not suitable commercial tools are on the GSA schedule contracts. GSA has negotiated schedule contracts for some commercial tools. Such contracts establish a baseline price and terms and conditions which individual agencies can use.

---

1/Object module is the output of the compiler, which translates the COBOL source statements written by the programmer to an intermediate machine-language object module.

2/Load module is the output of the linkage editor which links together the object code produced by the compiler and other programs and resolves references. It is ready for execution.

3/Link editing is the activity of the linkage editor in producing a load module.

INTERACTION WITH THE USERS OF APPLICATIONSFunctional users

The functional users (for example, the payroll department is the functional user of a payroll program) must be involved in the performance improvement process at, at least, two points: (1) the very beginning and (2) the turnover of an improved cheaper-to-operate version to production. At the very beginning, the users can supply information about cost and processing frequency as well as information about the production data and files--both what is typical or very common and what sort of anomalies have been encountered. The users must be involved in turnover because they must be satisfied that the revised cheaper-to-operate application still gives the same results over the same domain of input data as the original application.

Separate performance analysts and application programmers/analysts

If the group responsible for performance improvement work is separate from the applications programmers/analysts, considerable tact must be exercised and a cooperative spirit developed. Otherwise, there is an implication that the applications programmers are "unskilled, poor programmers because the performance group was able to make their program run twice as fast." Concerning this implication, the following should be kept in mind.

- Applications analysts/programmers have different skills. They must, for example, know a good deal about the application whereas performance persons need not.
- The performance group can educate the applications programmers to select alternatives that are more efficient. Some of these alternatives are as easy to use as less efficient ones.
- Applications programmers/analysts often have a backlog of user functional requests and don't have time for performance work.

SUMMARIES OF WHAT WE  
FOUND AT THE SITES VISITED

	<u>Page</u>
Sites at which we reviewed documentation of COBOL improvements	46
Sites at which we completed COBOL resource reduction experiments	50
Sites at which we discussed Government-wide aspects of COBOL performance improvement	55

SITES WE VISITEDINSTALLATIONS AT WHICH WE REVIEWED DOCUMENTATION  
OF COBOL PERFORMANCE IMPROVEMENT

We visited three Federal sites that we knew had steady efforts to reduce resources consumed by their COBOL applications. We wanted their thoughts on management of the work, methods used, and results.

Site 1: Housing and Urban Development  
Office of ADP Systems Development  
Washington, D.C.

This site had one UNIVAC 1100/81 and two UNIVAC 1108s. About 97 percent of the estimated 3,320 hours of yearly CPU use is consumed by COBOL applications. Many of the COBOL programs were still in ANSI 1968 standard COBOL when we visited but are being converted to ANSI 1974 COBOL. A paper published by a HUD representative 1/ discusses the agency's efforts and illustrates strikingly the 20/80 phenomenon: For a group of 78 systems executed on one mainframe at HUD, one system (1/78) consumed 20 percent of the total production CPU time by itself; eighteen systems (18/78--about one-fifth) consumed 82 percent of the CPU time--this is almost exactly 20/80 proportions.

HUD has a formal Standards and Quality Control Group whose responsibilities include "\* \* \* procedures to upgrade the efficiency of operational systems as well as procedures to enhance the reliability of new systems." The group is also responsible for performance improvement. The group uses system accounting data--aggregated into production utilization reports--to identify expensive applications and the COBOL Instrumentation Processor 2/ to collect execution counts and timings for programs of interest. Some hand analysis is done to evaluate input/output alternatives, and the SCORE package is used to generate COBOL programs to extract test data from production files. We analyzed HUD results from performance improvement of five systems to obtain a net benefit. Our results are shown in detail in chapter 2 of this report.

HUD representatives told us that their present level of effort will continue for the foreseeable future.

---

1/Grossman, "A Procedure to Review and Improve the Operational Efficiency of Production Systems," Proceedings of the Joint NBS/ACM Symposium, June 19, 1980.

2/Developed in the Department of the Navy.

Site 2: U.S. Army Computer Systems Command  
Ft. Belvoir, Va.

This site develops Standard Army Management Information Systems which are run at multiple sites and which include Honeywell, UNIVAC, and IBM computers. USACSC has an ongoing concern with performance improvement which is required by Army Regulation AR-10-9: But USACSC is motivated by the visibility of its centralized mission and by the fact that an improvement in a standard system is multiplied by the number of sites at which it runs.

USACSC uses a number of tools and techniques for reducing the machine resources consumed by COBOL applications: (1) review by a third party who is independent of the developer, (2) testing application systems in a test-bed site before wide release, and (3) using automatic optimizers and fine-tuning after a system has been in widespread production for 6 months or more. We were told that performance considerations are always part of development and qualification testing.

We were told that various vendor products and techniques have been used as appropriate and that significant and worthwhile reductions in machine resources consumed by USACSC applications have been achieved.

USACSC plans to continue its present activity in this area.

Site 3: House Information Systems  
Committee on House Administration  
House of Representatives  
Washington, D.C.

This site uses IBM-software-compatible computers with both batch and on-line applications, the latter running under Customer Information Control System. <sup>1/</sup> We visited Housing Information Systems because we had been told that they use automatic optimization heavily.

House Information Systems representatives showed us internal documentation of their benchmarks of two products of the CAPEX Corporation--OPTIMIZER II and its replacement, OPTIMIZER III. The representatives stated that OPTIMIZER III is the default that a programmer gets with a development compile, that this is done to ensure that programs put into production have been improved with the OPTIMIZER, and that OPTIMIZER III has recovered very significant resources. They felt that "dollar savings" would be more relevant if an outside timesharing service were used. Then, a reduced CPU time consumption would be directly visible as a lesser charge to run the COBOL program. They also stated that in their environment of short time frame user requests and frequent functional changes to programs, optimization by hand analysis, retesting, and the like would not be feasible.

---

<sup>1/</sup>Customer Information Control System, an IBM software product.

SITES AT WHICH WE COMPLETED COBOL RESOURCE  
REDUCTION EXPERIMENTS

We completed experiments at several Federal sites that use COBOL heavily. We wanted to demonstrate and verify the potential of different techniques and to satisfy ourselves that the general procedure is widely applicable. Several of our improved versions are now in actual production at their respective installations.

Site 1: Defense Mapping Agency  
Brookmont, Maryland

This site had a 15-year-old Burroughs B3500 computer with 235 of its 350 production programs written in COBOL with a compiler conforming to the 1968 COBOL standard. Before our visit, applications improvements had been limited to system flow improvements, such as combining programs to reduce the number of temporary files and changing processing from sequential to random. Also, the agency used the COBOL COPY facility to allow the repeated use of identical code by many programs, which improved programmer productivity. The agency was already planning a conversion when we arrived.

We used the monthly workload report--produced by a Burroughs product named LOGGER--and conferences with Defense Mapping Agency staff to identify two programs for experiment. To get execution counts, we used a software tool named COBTRAK which embeds additional COBOL statements in a COBOL program of interest so that when it is recompiled and executed, counts of how many times each paragraph executes will be displayed. COBTRAK is a GAO-enhanced rewrite of a program found at an installation we visited during our earlier software technology project, a report on which was published in April 1980. 1/

One of the programs, YDDC00, was the second largest CPU time user among the major COBOL applications. It processed a large file organized into master records and subrecords. About 40,000 master records organized by stock number could each have up to 100 subrecords representing customers interested in that stock number. A transaction against a master record could require actions on every subrecord. We tested this program with 12 transaction records. COBTRAK execution count data showed that these 12 transactions caused 54,000 accesses to master records and 1.4 million accesses to subrecords. Further analysis showed that this system needed redesign for a database approach which was not supported by the old Burroughs 3500--and was already planned for the proposed replacement. Thus, it was not feasible to rework YDDC00 in its present environment.

---

1/"Wider Use of Better Computer Software Technology Can Improve Management Control and Reduce Costs," FGMSD-80-38, Apr. 29, 1980.

The other program, YAVK20, was one of over 200 COBOL programs that used a standard print routine. The standard print routine was copied into COBOL programs with the COPY verb so that programmers would not need to write their own printing logic. Agency staff indicated that they thought it was inefficient. They also indicated that, since it is written in COBOL, the standard print routine would be carried over to the proposed replacement computer with little or no change. We were able to demonstrate three versions of our improved print routine--one which was optimized for CPU time, one for memory, and one which improved both CPU and memory. Which to choose and realized savings of each depended on the processing (for example, many or few lines of print) and whether the program included the COBOL SORT verb. Our improvements to YAVK20 yielded a 10-percent reduction in the CPU time per execution and amounted to 2 hours and 36 minutes of CPU time saved per year--about \$1,300 worth at local charge rates.

Agency officials indicated that they would do performance improvement on their replacement computer after conversion. We later inquired and were told that they had indeed converted the standard print routine to their new computer and that they will apply our standard print routine improvements to over 100 programs. While we cannot calculate the resource consumption reduction from this improvement, we are confident that it will be significant.

Site 2: U.S. Army Management Systems Support Agency  
Pentagon, Washington, D.C.

This site has IBM computers. About 60 percent of its production programs are COBOL. The COBOL programs for which we were able to obtain data consumed 700 CPU hours yearly--over \$2 million worth at the local rate. Data for 696 COBOL programs that were run during March 1980 graphically illustrate the 20/80 rule--the worst one-fourth (174) of the programs consumed 13 CPU hours while the other 522 programs together consumed 61 CPU minutes and 43 seconds. Before we visited the site, very little had been done about the machine resource consumption of the COBOL applications. The Director of USAMSSA said he could not be concerned with machine efficiency because of an enormous backlog of user-requested changes and because his computer specialist staff was 30 percent under strength. A commercially available tool--COTUNE II--was installed on the system but was used very little.

USAMSSA management suggested several programs for test purposes. Another program was discovered because USAMSSA systems programmers referred a user complaint to us. We used several tools and methods at USAMSSA--system log data, utility programs summarizing that data, the commercial COTUNE II analysis aid, utility programs that reported reorganization criteria for indexed files, the commercial OPTIMIZER III automatic optimizer and analysis aid (in a demonstration arranged by us), and interviews with maintenance programmers.

We achieved several improvements. One small application cost about \$160 per year to operate, and the maintenance programmer said he thought the output was no longer being used. We suggested to USAMSSA management that they get user agreement to stop running it. Also, we used the COTUNE II tool to aid hand analysis of a program named SEQFORCR. We reduced its CPU time consumed by 17 percent with better table handling and reducing its core requirement by specifying no alternate area for its parameter card file. We also inserted code to detect the absence of the parameter card after we had demonstrated that the program would run anyway--giving incorrect results--if its parameter card were left out.

We also made an improvement to a program named NEWFILEB. We reorganized its indexed sequential files--providing a larger prime data area--and cut the elapsed time from 7 hours 15 minutes to 1 hour 56 minutes and the CPU time from 8 minutes 36 seconds to 1 minute 38 seconds--reductions of three-fourths and four-fifths, respectively. We estimated \$2,764 annual savings in CPU time alone from these improvements. We also asked the CAPEX Corporation to demonstrate its product, OPTIMIZER III, on three programs. OPTIMIZER III succeeded in reducing the memory size of the three programs by 13, 16, and 23 percent.

We briefed USAMSSA programmers, analysts, and management on the various improvement possibilities. USAMSSA management reiterated their staffing problem and expressed interest in the CAPEX OPTIMIZER III because it offers improvement without labor.

Site 3: Navy Regional Data Automation Command  
Norfolk, Va.

This site has both UNIVAC and Burroughs computers. Our experiments were done on the UNIVAC 1100, model 40. About 1,400 of 1,652 production programs at NARDAC are in COBOL. Before we visited the site, its parent Navy Data Automation Command had issued instructions (in April 1979) to improve UNIVAC 1100 applications. This improvement was motivated by both resource usage and reliability concerns.

We used two tools to identify programs that consumed large amounts of resources. Those tools were the Resource Utilization System, developed by the Navy, and the Log Analysis Statistics, Summary, and Other Program, developed by UNIVAC. After we had selected two programs we used a third tool, the COBOL Instrumentation Processor, to aid our work on the individual programs. The COBOL Instrumentation Processor modifies the object module prepared from the COBOL source so that, when the program is executed, information is captured about how many times each part of the program executed and how long it took. Such information focuses modification effort on high cost parts of the program and verifies the improvement, if any, from a given modification. The COBOL Instrumentation Processor was also developed by the Navy.



We identified two programs that were heavy users of CPU time and modified their source code, using more efficient table handling and more efficient arithmetic than were used in the existing versions. We reduced by 40 percent the CPU time consumed by each program. At the present number of production runs per year, we estimated that our improved versions would save about 13 CPU hours in the first year of operation--\$650 at the local charge rate. We believe that further savings from improvements to other applications can be achieved.

We were told that the optimization directed by the Naval Regional Data Automation Command is underway, that programmers have been trained, and that all Navy Regional Data Automation Command locally written programs will be reviewed for optimization potential on a time-available basis.

Site: 4 Automatic Data Processing Division  
Bureau of Mines/Federal Center  
Denver, Colorado

This site has a Burroughs 6750 computer with 852 of its 1,032 production programs written in COBOL. The COBOL compiler (Burroughs 4.5) conforms to the ANSI 1974 standard. Before we arrived, the site had gone through an extensive, 3-year optimization effort under the leadership of a very skilled person who retired early in 1979. It was the opinion of this person--documented in a memo--that much of the worthwhile optimization was completed by the time he retired.

We used the system job summary to identify expensive programs and to compare the costs of several versions. The COBOL compiler includes two excellent tools in the form of compile options: ANALYZE and STATISTICS. The ANALYZE option gives messages about the inefficiencies it discovers during compilations, including potential excessive paging. The STATISTICS option causes the compiler to accumulate execution counts and timings for each paragraph of the source program.

Our work on the JA318F program showed that most of its time (92-95 percent) was spent in communicating with a Burroughs-supplied Data Base Management System. Although we thought that becoming involved with the Data Base Management System was not feasible, we did accomplish minor improvements in arithmetic by changing data types. We did this easily and quickly, but the accomplishment was not significant. With the LF7440 program, we demonstrated a significant potential for improvement by controlling the paging of the program. Burroughs' "virtual memory" can be controlled at the COBOL source language level by using the vendor-unique semantics of the SECTION header and the SEGMENT-LIMIT clause. Burroughs' ANALYZE option gives messages about the program's paging behavior. The program's PROCEDURE DIVISION can then be reorganized to reduce the paging.

However, the practical value of the improvement depends greatly on the environment, that is, how many other programs are contending for memory at the same time as the program of interest. The fact that LF7440 was well-structured made it easy to reorganize without fear of unforeseen side effects. On the other hand, the third program, LT5050, was so poorly structured (14 ALTERS, many GO TO's, etc.) that we judged it infeasible for optimization because prohibitive amounts of analyst labor and retesting would have been required to verify an improvement.

The director of the installation indicated he felt that periodic performance improvement is appropriate for a site the size of his--that the full-time continuous effort of even one highly skilled analyst is not justified. He also felt that contractor or temporary-position improvement analysts would be feasible if in-house skills were not available.

Site 5: Corps of Engineers  
North Atlantic Division  
Norfolk, Virginia

This site has a Honeywell 6120 computer. About 90 percent of its production applications are COBOL. When we visited, the majority of these were compiled with the Honeywell compiler which conformed to the 1968 ANSI COBOL standard. The new 1974 standard compiler had seen little use. Also, there seemed to be little awareness of performance improvement.

Data center personnel identified two programs as good candidates for improvement. The Honeywell documentation of the two COBOL compilers (1968 standard and 1974 standard) showed that the newer compiler used a different data management routine--the Universal File Access System--from that used by the older 1968 compiler.

We converted two programs to compile and execute under the 1974 compiler and achieved a 20-percent reduction in CPU time compared to the 1968 versions. We believe that the speed improvement is at least partly due to the interaction of the access system with the new compiler because the two programs were heavily input/output-oriented. The conversions were not difficult. Most changes needed for recompilation were due to different requirements for using the SIZE option. We also identified a general installation resource constraint: Insufficient disk storage required frequent copying of files to tapes to free disk space needed for other processing.

The director of the data center said that center officials plan to upgrade all locally maintained COBOL applications to ANSI 1974 COBOL starting with formal training in ANSI 1974 COBOL for all data center and district programmers. He also said that more disk devices had been ordered.

SITES AT WHICH WE DISCUSSED GOVERNMENT-WIDE  
ASPECTS OF COBOL PERFORMANCE IMPROVEMENT

Three organizations have Government-wide missions relating to COBOL applications performance improvement: FEDSIM, NBS, and the GSA compiler testing center. We visited them and discussed the subject. Much of what they told us is reflected in our discussion in chapters 2 and 3 of this report and in our provisional checklist.

Site 1: Federal Computer Performance Evaluation  
and Simulation Center  
Department of the Air Force  
Washington, D.C.

FEDSIM was established to develop and maintain programs, models, and techniques for simulating and analyzing automatic data processing systems and equipment for all Federal agencies. These tools are applied to various data processing systems and environments to improve ADP equipment selection, utilization, and performance.

A member of the FEDSIM technical staff said that their work is request work. He said they get very few requests for reducing the machine costs of applications (in any language) because management in the various agencies does not see machine costs as their most severe problem.

However, he did discuss two Federal sites that FEDSIM has worked with. One site operated a mix of COBOL and FORTRAN applications. The COBOL applications were originally written efficiently by professional programmers and little gain was made. On the other hand, the FORTRAN programs were written by engineers and analysts--not professional programmers--and significant resource savings were seen. The second site obtained significant COBOL improvements by "run stream optimization"--combining small programs into larger ones and eliminating redundant files.

The thoughts of the member of the FEDSIM technical staff on the management of COBOL performance improvement are included in our provisional checklist. He thought that "operational" problems--such as a new computer being saturated too quickly by blindly converted programs, or output delayed by elapsed times that were too long--were stronger motivations for performance improvement than cost savings. He also said that he felt Government-wide guidance is needed, perhaps in the form of a guidebook from NBS on the efficient and effective use of COBOL. He said that this guidance is needed to increase both the programmers' awareness of the performance implications of their work as well as their knowledge of better ways to use COBOL.

Site 2: General Services Administration  
Office of Software Development  
Falls Church, Va.

The Office of Software Development includes the Federal Compiler Testing Center and the Federal Conversion Support Center. The former validates COBOL compilers for conformance to the Federal Information Processing Standard 21-1 1/ and the latter assists agencies that are converting their applications programs.

The head of the compiler testing center said that, in his experience, performance improvement work need not interfere with adherence to the COBOL language standard. He said that significant reductions in applications' operating costs can often be achieved without using nonstandard COBOL language extensions. Concerning the use of vendor-unique COBOL features, he said:

--All such use should be justified by significant cost savings.

--All such use should be well documented.

--Vendor extensions which do not affect the user's stored data or computational results are much easier to take out later. An example he gave is that of COBOL language features that communicate with the vendor's data management routines, such as a request for a file index to be copied into core--IBM's APPLY CORE-INDEX. Other vendor-unique extensions, such as packed decimal data, do affect the user's stored data and can make conversion of the programs and files to another brand of computer much more difficult.

Concerning programmer training, he feels many COBOL programmers know one-fourth or less of the COBOL vocabulary and know little about how to use COBOL effectively and efficiently.

The head of the Office of Software Development and the head of the compiler testing center said that a Federal guidance document to show programmers the effective and efficient use of COBOL would be very helpful. They said, and we agree, that such a book should include concrete examples relevant to tasks that working COBOL applications programmers commonly encounter. They said that their organization would like to collaborate with the National Bureau of Standards on a project to produce such a document.

---

1/Compiler validation as done by the GSA for COBOL tests whether or not a compiler includes the language elements of the standard and that those standard elements work the way the standard says. A compiler thus validated may also include nonstandard extras called extensions.

Site 3: U.S. Department of Commerce  
National Bureau of Standards  
Institute for Computer Science and  
Technology  
Washington, D.C.

According to the NBS program plan, the following responsibilities assigned to the Secretary of Commerce under Public Law 89-306 have been delegated to the Institute for Computer Science and Technology:

- To provide Federal agencies with scientific and technological advisory services for ADP and related systems.
- To develop and recommend the establishment of uniform Federal ADP standards and to undertake necessary ADP research.

The NBS standards program includes a standard for the COBOL language itself and software quality standards of good practice that have been shown to lead to better, more efficient, and more easily maintained software systems.

Much of what the NBS representatives said about the management of performance improvement is reflected in chapters 2 and 3 and appendix I of this report.

One of the NBS representatives said that it is often very difficult to calculate a true dollar value for benefits gained. For example, if the procurement contract for equipment is fixed, reducing an application's CPU time will not immediately reduce dollar outflow. We explained to him that we had come to think in terms of recovered capacity that can be used for other purposes and that eventually reduces costs because it allows a procurement to be deferred.

Both NBS representatives stated that they felt that other software quality factors must receive more attention than machine resource consumption and cited their publication on software management. <sup>1/</sup> They pointed out that the impact of incorrect software on the user's supported application can be far greater than the cost of machine resources consumed, although it is difficult to quantify. Both agreed that many COBOL programmers get little formal training, are expected to learn on the job, and have little concern for machine costs.

Concerning Government-wide guidance, the NBS representatives pointed out that while they do not publish specific guidance on

---

<sup>1/</sup>NBS Special Pub: 500-11, op. cit., p. 30.

using COBOL for applications, many of their existing publications 1/ contain general principles that will yield more accurate, efficient, and effective programs in COBOL as well as in other languages. They also pointed out that there are a number of commercial books on COBOL usage and that it is not clear that programmers who do not read those will be any more likely to read a COBOL book with an NBS cover on it. We pointed out that NBS felt it worthwhile to publish a book on FORTRAN usage (Handbook 131) before FORTRAN had even become a Federal standard language and that we felt COBOL deserved separate treatment because its use is so widespread.

---

1/Including NBS Special Publication 500-56, "Verification, Validation, and Testing for the Individual Programmers."

LIST OF REFERENCESU.S. General Accounting Office Publications

- "Federal Agencies' Maintenance of Computer Programs: Expensive and Undermanaged," AFMD-81-25, Feb. 26, 1981.
- "Conversion: A Costly, Disruptive Process That Must Be Considered When Buying Computers," FGMSD-80-35, June 3, 1980.
- "Wider Use of Better Computer Software Technology Can Improve Management Control and Reduce Costs," FGMSD-80-38, Apr. 29, 1980.
- "Contracting for Computer Software Development--Serious Problems Require Management Attention to Avoid Wasting Additional Millions," FGMSD-80-4, Nov. 9, 1979.
- "Computer Performance Evaluation (CPE): An Auditor's Introduction," Nov. 1979.
- "Auditing Computer Based Systems," March 1979.
- "The Federal Information Processing Standards Program: Many Potential Benefits, Little Progress, and Many Problems," FGMSD-78-23, Apr. 19, 1978.
- "Illustrative Accounting Procedures for Federal Agencies: Guidelines for Accounting for Automatic Data Processing Costs," Federal Government Accounting Pamphlet Number 4, U.S. GAO, 1978.
- "The Federal Software Exchange Program--A Small Step in Improving Computer Program Sharing," FGMSD-78-11, Jan. 13, 1978.
- "Millions in Savings Possible in Converting Programs from One Computer to Another," FGMSD-77-34, Sept. 15, 1977.
- "A Working Glossary of Computer Software Terms," Sept. 15, 1977.
- "A Selected Bibliography on Computer Software Conversion," Sept. 15, 1977.
- "Auditing Computers with a Test Deck," Dec. 1975.
- "Tools and Techniques for Improving the Efficiency of Federal Automatic Data Processing Operations," B-115369, June 3, 1974.
- "Opportunity for Greater Efficiency and Savings Through the Use of Evaluation Techniques in the Federal Government's Computer Operations," B-115369, Aug. 22, 1972.

National Bureau of Standards Publications

- "Synopsis of Interviews from a Survey of Software Tool Usage," Herbert Hecht, NBS Internal Report 81-2388, Nov. 1981.
- "Final Report: A Survey of Software Tools Usage," Herbert Hecht, NB Special Publication 500-82, Nov. 1981.
- "Proceedings of the NBS/IEEE/ACM Software Tools Fair," Raymond C. Houghton, Jr., Editor, NBS Special Publication 500-80, Oct. 1981.
- "NBS Programming Environment Workshop Report," Martha A. Branstad and W. Richards Adrion, Editors, NBS Special Publication 500-78, June 1981.
- "Validation, Verification, and Testing of Computer Software," W. Richards Adrion, Martha A. Branstad, and John C. Cherniavsky, NBS Special Publication 500-75, Feb. 1981.
- "Software Development Tools: A Reference Guide to a Taxonomy of Tool Features," NBS Letter Circular 1127, Feb. 1981.
- "Features of Software Development Tools," Raymond C. Houghton, Jr., NBS Special Publication 500-74, Feb. 1981.
- "NBS Software Tools Database," Edited by Raymond C. Houghton, Jr. and Karen A. Oakley, NBSIR-80-2159, Oct. 1980.
- "Computer Performance Evaluation Users Group (CPEUG) 16th Meeting," Harold Highland, Editor, NBS Special Publication 500-65, Oct. 1980.
- "Conversion of Federal ADP Systems: A Tutorial," Joseph Collica, Mark Skall, Gloria Bolotsky, NBS Special Publication 500-62, Aug. 1980.
- "Final Report Software Tool Taxonomy," prepared by Donald J. Reifer and Harold A. Montgomery, prepared for NBS under contract NB79SBCA0273, June 1, 1980.
- "Using ANS FORTRAN," Gordon Lyon, Editor, NBS Handbook 131, Mar. 1980.
- "Validation, Verification, and Testing for the Individual Programmer," Martha A. Branstad, John C. Cherniavsky, W. Richards Adrion, NBS Special Publication 500-56, Feb. 1980.
- "Computer Performance Evaluation Users Group (CPEUG) 15th Meeting," James E. Weatherbee, Editor, NBS Special Publication 500-52, Oct. 1979.
- "Technology Assessment: ADP Installation Performance Measurement and Reporting," NBS Special Publication 500-53, U.S. Department of Commerce, Sept. 1979.



- "Guidelines for Documentation of Computer Programs and Automated Data Systems for the Initiation Phase," FIPS PUB 64, Aug. 1, 1979.
- "Guide to Major Job Accounting Systems: The Logger System of the UNIVAC 1100 Series Operating System," National Bureau of Standards Special Publication 500-43, Dec. 1978.
- "Computer Performance Evaluation Users Group (CPEUG) 14th Meeting," James E. Weatherbee, Editor, NBS Special Publication 500-41, October 1978.
- "COBOL Instrumentation and Debugging: A Case Study," Gordon Lyon, NBS Special Publication 500-26, Jan. 1978.
- "Computer Performance Evaluation Users Group," Edited by Dennis M. Conti and Josephine L. Walkowicz, NBS Special Publication 500-18, Sept. 1977.
- "Software Tools: A Building Block Approach," I. Trotter Hardy, Belkis Leong-Hong, and Dennis W. Fife, NBS Special Publication 500-14, Aug. 1977.
- "Computer Software Management: A Primer for Project Management and Quality Control," Dennis W. Fife, NBS Special Publication 500-11, July 1977.
- "Guideline On Computer Performance Management: An Introduction," FIPS PUB 49, May 1, 1977.
- "Appraisal of Federal Government COBOL Standards and Software Management: Survey Results," Donald R. Deutsch, NBSIR, 76-1100, Aug. 1976.
- "Guidelines for Documentation of Computer Programs and Automated Data Systems," FIPS PUB 38, Feb. 15, 1976.
- "Aids for COBOL Program Conversion," FIPS PUB 43, Dec. 1975.
- "COBOL," FIPS PUB 21-1, Dec. 1, 1975.

#### Other Government Publications

- "Software Improvement--A Needed Process in the Federal Government," Report OSD-81-02, Office of Software Development, ADTS, GSA, June 3, 1981.
- "Management Guidance for Developing and Installing an ADP Performance Management Program," GSA, Nov. 1978.
- "Performance Improvement ASCII COBOL and the Run Stream," Forum Proceedings Fort Collins Computer Center, Nov. 1977.
- "American National Dictionary for Information Processing," American National Standards Committee X3-Computers and Information Processing, Sept. 1977.

"Computer Program Optimization," Annette J. Krygiel, Defense Mapping Agency Aerospace Center, Apr. 1974.

"Structured Programming Using COBOL 1974 (ANSI COBOL)," FADPUG Presentation by George Baird, March 13, 1974.

"Public Law 89-306," 89th Congress, H.R. 4845, Oct. 30, 1965.

#### Nongovernmental Publications

"Software Performance Viewed A Design Problem," Rita Shoor, ComputerWorld, Page 14, Dec. 15, 1980.

"A Procedure to Review and Improve the Operational Efficiency of Production Systems," Robert A. Grossman, June 19, 1980, Proceedings of Joint NBS/ACM Symposium on Pathways to System Integrity.

"Quality Assurance Tools," Robert W. Shirey, ComputerWorld, pp. 31-49, May 19, 1980.

"Tools and Techniques," EDP PERFORMANCE MANAGEMENT HANDBOOK, pp. 4.650.1-4.650.4, Mar. 1980.

"The Art of Software Testing," G.J. Myers, New York, Wiley, 1979.

"The COBOL Environment" Crawford and Grauer, Englewood Cliffs, N.J. Prentice-Hall, 1979.

"Controlled Resource Management Through Computer Performance Evaluation," Edited by C. M. Edwards, III, John C. Kelly, Charles Ross, Computer Measurement Group, Inc., Dec. 1979.

"The Next COBOL Standard," Robert Fried and Robert McKenzie, Datamation pp. 175, 178, 180, Sept. 1979.

"Management Perspectives on Programs, Programming and Productivity" Presented by Robert C. Kendall, GUIDE 48, 1979.

"COBOL--The 1980 Standard: A Preview," Auerbach Publishers pp. 1-12, 1978, Computer Programming Management.

"Making the Most of Performance Monitors," William B. Engle, Computer Decisions, p. 50, Nov. 1978.

"Improving Software Performance at the Social Security Administration," Amr A. El-Sawy, Timothy L. Oliver, Eric D. Siegel, MITRE Technical Report MTR-8044, Sept. 1978.

"Execution Time Optimization of COBOL 5 Programs," Sunhyvale, Calif., Development Division, Control Data Corporation, Sept. 15, 1978.

- "A Critical Assessment of EDP Objectives," McCaffery, Seligman, and von Simson, Inc., Sept. 1978.
- "A Survey of EDP Performance Measures," Joseph R. Matthews, Government Data Systems, pp. 29-32, Jul./Aug. 1978.
- "COBOL Optimization Techniques" Judith Stevens Budney, Seventeenth Annual Technical Symposium, NBS, June 15, 1978.
- "The Impact of Program and Programmer Characteristics on Program Size," Earl Chrysler, National Computer Conference, AFIPS Conference Proceedings, pp. 581-87, June 5-8, 1978.
- IBM VS COBOL for OS/VS (Compiler & Library), Second Edition, May 1978, IBM Corporation, Armonk, N.Y.
- "Optimizing Program Quality and Programmer Productivity," Capers Jones, IBM Corp. SHARE 50, Mar. 7, 1978.
- "A Method for the Time Analysis of Programs," S.L. de Freitas and P.J. Lavelle, IBM Systems Journal pp. 26-38, 1978.
- "EDP Effectiveness Evaluation," Corydon D. Hurtado; Journal of Systems Management, pp. 18-21, Jan. 1978.
- "Gaining An Awareness of the Performance of COBOL Programs," Paul J. Jalics, Computer Measurement Group Proceedings, pp. 61-65, 1978.
- "Software Metrics," Tom Gilb, 1977, Winthrop Publishers, Inc.
- "Penguin Dictionary of Computers," Anthony Chandor, John Graham, Robin Williamson, Second Edition, 1977.
- "Software Tuning, Not Upgrade, Urged for 370 Sites" Don Leavitt, ComputerWorld p. 13, Oct. 4, 1976.
- "Array Handling in COBOL Compilers," M. H. Williams and A. R. Bulmer, Software Practice and Experience, pp. 469-474, 1977.
- "The Effect of COBOL Efficiency Techniques on Computer Run Costs," Nancy V. McGuire, Sixteenth Annual NBS/ACM Technical Symposium, Systems and Software, June 2, 1977.
- "COBOL Optimization and Flowcharting" Mary W. Headrick, FOCUS-17 Conference, May 23-26, 1977.
- "Improving Performance the Easy Way," Paul J. Jalics, Datamation, pp. 135-137; Apr. 1977.
- "COBOL Tuning in VS" Session Report, SHARE 48, Volume II, Mar. 1977.



**UNITED STATES DEPARTMENT OF COMMERCE**  
**The Inspector General**  
Washington, D.C. 20230

December 10, 1981

Mr. Henry Eschwege  
Director, Community and Economic  
Development Division  
U. S. General Accounting Office  
Washington, D. C. 20548

Dear Mr. Eschwege:

This is in reply to your letter of November 6, 1981,  
requesting comments on the draft report entitled  
"Improving COBOL Applications Can Recover Significant  
Computer Resources".

We have reviewed the enclosed comments of the Acting  
Assistant Secretary for Productivity, Technology and  
Innovation for the Department of Commerce and believe  
they are responsive to the matters discussed in the  
report.

Sincerely,

A handwritten signature in cursive script, appearing to read "Sherman M. Funk".

Sherman M Funk  
Inspector General

Enclosure



**UNITED STATES DEPARTMENT OF COMMERCE**  
**The Assistant Secretary for Productivity,**  
**Technology and Innovation**

Washington, D C 20230  
(202) 377-3111

**NOV 25 1981**

Mr. W. D. Campbell  
U.S. General Accounting Office  
441 G Street, N.W.  
Washington, D.C. 20548

Dear Mr. Campbell:

Thank you for the opportunity to comment on the draft report entitled "Improving COBOL Applications Can Recover Significant Computer Resources." Improvement of high-use Federal computer application programs written in the COBOL programming language can be a worthwhile and cost-saving objective, and the report wisely cautions against optimization at the expense of language standards (p.22).

We do not concur at this time in the report's recommendation that "The National Bureau of Standards (NBS) should publish guidance on the effective and efficient use of COBOL for applications." Optimization of application software can occur at several levels in a computer system, from the microcode that executes machine language instructions to the level at which source programs migrate from one hardware configuration to another. In its planning to develop most urgently needed standards and guidelines, the Institute for Computer Sciences and Technology (ICST) at NBS has chosen to address the three highest levels of optimization where the return on agency effort is greatest: inter-system portability (language standards), total system performance, and application software life-cycle management. At the present time, this three level, top-down approach to the problem of sub-optimal application software has higher priority in allocating ICST's appropriated resources than developing guidance in the use of specific source languages, such as COBOL. If the opportunity to develop this product on a reimbursable basis should arise, however, we would expect ICST to review its staffing requirements and reconsider such an undertaking. A supplementary list of ICST products that are broadly applicable to the development and maintenance of COBOL programs, in addition to those listed in Appendix III of the draft report, is attached.

The report questions ICST's reluctance to develop a handbook for COBOL when it had already published one for FORTRAN (pp. 14 and 56). Some background may help to explain the reason. The FORTRAN handbook was to have been part of a series of handbooks for all major programming languages, including COBOL, to be written by leading experts in the field. Only one such expert, a FORTRAN specialist who had already developed most of the raw materials on his own time, was willing to take on the task for the amount that ICST was able to allocate to it. Although this one product has been useful and well-received, ICST concluded that the language-by-language approach to improving software efficiency was not feasible for a program of its size.

We propose the following specific changes in the draft report:

1. Delete or modify the sentence beginning on page 13, line 8 that reads: "An NBS official told us he felt that most COBOL programmers have the attitude that getting the program work any way they can is satisfactory."

This is not the opinion of ICST. It would be more accurate to say that many COBOL programmers often have to settle for producing a program that works, but that the result is often not satisfactory to the conscientious programmer.

2. Delete or modify the sentence beginning on page 14, line 19 that reads: "The NBS officials also said that it is not clear that guidance published by NBS would be read or followed any more than commercially published guidance."

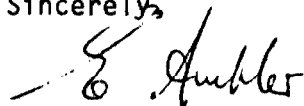
The implication is that government programmers neither read nor follow published guidance, and that the source of the guidance makes no difference. ICST does not agree with either implication. Programmers are eager for high quality guidance, and we have found that they tend to give ICST guidance documents close attention and respect.

3. Delete or modify the sentence on page 14, line 21 (and the similar sentence on page 56) that reads: "Our representative pointed out to the NBS representatives that NBS had already published specific guidance on using the FORTRAN language and if that was worthwhile, surely COBOL guidance would be also."

This is misleading in view of the background for the FORTRAN handbook summarized above.

In summary, this report is a worthwhile attempt to focus on the need for careful and continuous attention at the ADP installation level to the efficiency of application software.

Sincerely,



Ernest Ambler  
Acting Assistant Secretary

Enclosure

cc: Henry Eschwege

## Suggested Additional NBS Products

"Computer Performance Evaluation Users Group (CPEUG) 14th Meeting," James E. Weatherbee, Editor, NBS Special Publication 500-41, October 1978.

"Computer Performance Evaluation Users Group (CPEUG) 15th Meeting," James E. Weatherbee, Editor, NBS Special Publication 500-52, October 1979.

"Computer Performance Evaluation Users Group (CPEUG) 16th Meeting," Harold Highland, Editor, NBS Special Publication 500-65, October 1980.

"NBS Programming Environment Workshop Report," Martha A. Branstad and W. Richards Adrion, Editors, NBS Special Publication 500-78, June 1981.

"Proceedings of the NBS/IEEE/ACM Software Tools Fair," Raymond C. Houghton, Jr., Editor, NBS Special Publication 500-80, October 1981.

"Final Report: A Survey of Software Tools Usage," Herbert Hecht, NBS Special Publication 500-82, November 1981.

"Synopsis of Interviews from a Survey of Software Tool Usage," Herbert Hecht, NBS Internal Report 81-2388, November 1981.

"Features of Software Development Tools," Raymond C. Houghton, Jr., NBS Special Publication 500-74, February 1981.

"Software Development Tools: A Reference Guide to a Taxonomy of Tool Features," NBS Letter Circular 1127, February 1981.

"Aids for COBOL Program Conversion," FIPS PUB 43, December 1975.

"Validation, Verification, and Testing of Computer Software," W. Richards Adrion, Martha A. Branstad, and John C. Cherniavsky, NBS Special Publication 500-75, February 1981.



General  
Services  
Administration

Washington, DC 20405

---

DEC 1 1981

Honorable Charles A. Bowsher  
Comptroller General  
of the United States  
U.S. General Accounting Office  
Washington, DC 20548

Dear Mr. Bowsher:

In response to Mr. Horan's letter of November 9, 1981, enclosed are the General Services Administration's comments on the draft GAO Report "Improving COBOL Applications Can Recover Significant Computer Resources."

Sincerely,

Administrator

Enclosure



GSA welcomes the opportunity to review and comment on your draft report to the Congress entitled "Improving COBOL Applications Can Recover Significant Computer Resources." There cannot be enough emphasis placed on the importance of computer software to agencies' effectiveness. It was to this end that GSA formed the Office of Software Development to provide a central focus for the resolution of problems associated with agencies' computer programs. Your report supports our policy of encouraging agencies to undertake substantial software improvement programs as outlined in our monograph "Software Improvement - A Needed Process in the Federal Government." <sup>1</sup> Your report adds specific items of knowledge and guidance to this.

It is particularly good to notice you emphasize that projects to reduce computer resources consumed are only valid when such reductions repay their own cost without conflicting with such other software management objectives as-

1. preserving or improving the maintainability of the programs;
2. ensuring they meet the users needs; and
3. ensuring reliability, with defenses against problem situations.

Building and maintaining software is still a labor intensive process and, while labor costs are increasing annually, productivity has not significantly increased since the introduction of high level languages. Hardware per-unit costs meanwhile have plummeted and can be expected to continue to do so during this decade. Software is already the major ADP cost today, accounting for approximately two thirds of total ADP costs. Given the above trends, this proportion can only increase in the foreseeable future. Hence, software improvement strategies that substitute machine processing for programmer labor are to be encouraged and such contrary strategies as reducing computer resource consumption at the expense of maintainability are to be discouraged.

Of all fields, computer science is one where the current and projected shortfall of trained staff is very alarming and in considering software improvement projects it is necessary to recognize the constraints of finite human resources. It is not sufficient that this type of work generates enough reduction of resources consumed to offset its own labor and machine costs. It is necessary to consider lost opportunities and to identify and tackle the highest priority (and potentially largest pay-off) improvements rather than approach software improvement on a piecemeal basis. GSA is highly recommending agencies (particularly those with large systems) prepare a multi-year software improvement plan to ensure that priorities are correctly set.

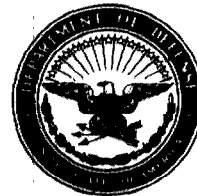
---

1. Report OSD-81-102, Office of Software Development, ADTS, GSA, June 3, 1981.

Finally, we would emphasize quality control and testing in conducting any software improvement program. While at one end of the spectrum simple optimization by use of an automatic optimizer would probably not require significant testing, improvements that include such items as elimination of intermediate files, file restructuring or program restructuring should not be undertaken without a testing plan, methodology, and strict quality control. Otherwise, faults will be introduced which will degrade the other software management objectives you mention of maintainability, user responsiveness and reliability.

Our Office of Software Development will follow your recommendation of working with the National Bureau of Standards (NBS) in publishing guidance on the effective and efficient use of COBOL applications and will also continue its program of providing further guidance and assistance to agencies in improving their software.

DEPARTMENT OF THE AIR FORCE  
FEDERAL COMPUTER PERFORMANCE EVALUATION AND SIMULATION CENTER (AFCC)  
WASHINGTON, D.C. 20330



27 November 1981

Mr. W. D. Campbell  
Acting Director, Accounting and  
Financial Management Division  
United States General Accounting Office  
Washington, D.C. 20548

RE: Your Letter dated November 6, 1981

Dear Mr. Campbell:

FEDSIM has no comment on the substance of the draft report "Improving COBOL Applications Can Recover Significant Computer Resources". We have not had extensive experience in the optimization of COBOL programs and thus do not have the background be able to characterize Government-wide practices in this area.

In this regard, I would appreciate it if you would clarify the point that Mr. McKenzie was informally contacted by your office as an individual and not as a "FEDSIM representative." Thus any views he expressed should not be construed as being based upon the "corporate" experience of FEDSIM as a whole.

Sincerely,

THOMAS GIAMMO  
Technical Director

\*U.S. GOVERNMENT PRINTING OFFICE : 1982 O-361-843/2069

(913630)

Vertical line of text on the left side of the page.

Vertical line of text on the right side of the page.

• - •

•

-----

-----

**AN EQUAL OPPORTUNITY EMPLOYER**

**UNITED STATES  
GENERAL ACCOUNTING OFFICE  
WASHINGTON, D.C. 20548**

**OFFICIAL BUSINESS  
PENALTY FOR PRIVATE USE, \$300**

**POSTAGE AND FEES PAID  
U. S. GENERAL ACCOUNTING OFFICE**



**THIRD CLASS**