Guidelines to software conversion

by PAUL OLIVER
Department of the Navy
Washington, DC

BACKGROUND

Data processing installations preparing to undergo a system transition are faced with several difficulties; prominent among these being the transition of the current workload to a new system.

A report by the General Accounting Office published in September 1977, states that the annual Federal Government cost of modifying computer programs to enable them to execute correctly on a computer different from the one for which they were originally devised is estimated at more than $450 million. Comparable industry-wide figures are not available, but it is reasonable to assume that the overall cost of software conversion is significant. Furthermore, this is a non-productive cost; conversion per se results in no direct improvement in an organization's ability to fulfill its mission.

Research and development efforts are under way at several universities and research laboratories to determine ways and means of producing portable software, i.e., software which is machine and configuration independent over a set of computer installations (e.g., see References 2 and 3). At the same time, industry is reacting to the problem in a variety of ways, including a softening of architectural differences (e.g., there are some half dozen IBM 370 'derivatives') and an improvement in emulation capabilities. Until such efforts bear practical fruits, data processing organizations will periodically be faced with the prospect of a software conversion effort. Such an effort is invariably faced with distaste and apprehension.

These are several reasons why system conversion is such a disruptive process. First of all, programmers must be shifted from their regular assignments to the conversion task. This is true whether or not an outside contractor is used to assist in the conversion. Proper conversion requires documentation, and old documentation is often found to be inadequate, even in well managed installations. Aggravating this condition is the fact that the programmers who originally coded the system are frequently no longer with the organization. It is also a sad fact that very few software development specifications include provisions for portability, even though techniques for reducing the impact of system changes on software are known and documented. Finally, conversion often will take place in conjunction with the implementation of a new system, thereby adding to the concomitant disruption.

Conversion is a many-faceted enterprise. An organization preparing itself for a conversion needs to consider not only the conversion process itself but also its management, the technical problems to be faced, the relationship of the conversion costs to the cost evaluation of the new systems being considered, the procurement of conversion support services, cost and time estimates, conversion alternatives (such as emulation), and training of personnel. A complete discussion of all of these topics is beyond the scope of a single paper. We will concentrate therefore on the description of the conversion process itself and the technical difficulties associated with it. The emphasis will be on what needs to be done rather than on how to do it. This emphasis reflects a value judgment on the part of the author; namely, that most of the ignorance regarding conversion has to do with the process itself. All too often organizations mistakenly liken conversion to development, fail to plan and prepare properly, and invariably are overly parsimonious in the allocation of resources to the conversion effort.

Before proceeding, the following definitions are provided in order to set the scope of this paper:

"Conversion: By conversions we mean any change made to a program or system of programs solely for the purpose of enabling such a program or system to execute correctly on a computer different from the one for which they were originally devised. Translation refers to a largely automated process of conversion in which the original programs themselves serve as adequate specifications for the new programs to be produced. Recoding is similar to translation except that the process is largely manual. Re-Programming refers to a conversion which may entail a system redesign (e.g., batch to on-line) but no significant functional redesign. Redesign refers to a conversion effort which involves functional redesign and is therefore akin to new development."

CONVERSION PROJECT OVERVIEW

Preparation

The first step in the preparation for conversion is the requirements analysis. A review of the planned differences between existing systems and the converted system is par-
particularly important if the language dialect being converted
to has new language modules (which may be fruitfully used
in the converted system), or major changes to input-output
modules. It will also be necessary to identify the degree to
which the compiler being used differs, in its implementation
of the language, from the standard specifications for that
language.

Tasks, schedules, resource requirements, and end prod-
ucts must be identified. Schedules and resource require-
ments are particularly difficult to gauge. It is generally a
good idea to obtain contractor support in preparing time and
resources estimates, since broad experience in conversion
is required in making such estimates. The review of existing
programs may reveal some which will require no conversion.
It is doubtful that there will be many of these. The rest must
be collected together with accompanying files and docu-
mentation, and placed in the hands of a conversion team.

Finally, the specifications of system changes must be de-
defined. Data file changes may be required; file and record
sizes, field contents, file organization, access keys, sort
keys, access methods, storage media, and labeling conven-
tions are all likely candidates for change. The conversion
will present an opportunity for some needed system restruc-
turing; programs to be combined, intermediate files elimi-
nated, or sort/merges which may be deleted if the restruc-
turing involves a shift from tape to DASD residence for
certain files. Also carefully specified must be any processing
logic changes which are necessitated by differing language
dialects.

Software tools must be identified and developed. Software
must be available to load data, copy programs, create ex-
tracted versions of test data, perform data and file conver-
sions, compare tests results for validity, and measure tests
for reliability. Procedures must be developed and controls
and quality assurance standards must be specified.

Programs must be collected in a uniform way to ensure
that the correct version (release) on every program is being
converted. The software indicated above may be used to
create test libraries and to control this step. A procedure for
maintenance change inclusion must be developed, and a
reference base for changes is hereby established.

Adequate test data must be prepared which will exercise
an acceptable portion of the converted programs. Each pro-
gram should have a set of known inputs which produces
known outputs in order to validate each program. Ideally,
unit test data can also be used for system testing.

Finally, all related materials must be collected. This in-
cludes program and system documentation (flowcharts, nar-
ratives, run-books, data layouts), inventories of files and
programs, source listings, program assemblies (listings), and
a directory of every item’s physical location.

Production

As subsystems become ready for actual conversion the
translation process begins. This is true even if systems are
to be eventually modified. That is, the conventional wisdom
is that program translation (i.e., a one-for-one, or close to
it, conversion) should precede any modification. This is
done to avoid intermingling, and thereby compounding any
translation errors or effects with modification errors or ef-
effects. The success of the conversion will be closely related
to the adequacy of the controls which are applied during the
production stage. Controls must be established for the re-
cipe, handling, and distribution of all materials, for the
 copying and analysis (to ensure the correctness of the copy
process) of program tapes; and for the definition and use of
job control language programs.

The translation process itself will be partly automated.
Many features of programming languages lend themselves
to automatic translation, using commercially available or in-
house developed utilities. Unfortunately, a large portion of
the input-output coding will have to be hand-translated, and
in some cases the process will necessarily have to be closer
to modification than translation.

Throughout unit and system tests should follow the tran-
slation phase (and will have to be repeated after the modifi-
cation phase, if any). It is generally advisable to desk-trace
the programs in a gross way, i.e., through job control, housekeeping, and initial input statements. A monitor which
intercepts and analyzes abnormal terminations would be an
extremely useful tool during testing. The monitor should
be capable of displaying the instruction causing the abnormal
termination, the data being processed at that time, and of
providing snapshots of selected data/program areas. Also
useful would be a file-compare utility to determine the va-
idity of the outputs produced by the translated program and
a monitor which could recognize units of untested code.
Once a translated system has been successfully tested any
required modifications can take place. These may include
system restructuring (combination of common subroutines,
sort/merge utilities, etc.), changes in logic, and changes in
data files.

Finally, the entire process must be thoroughly and care-
fully documented. The precise form of the documentation
will depend on the installation's standards, but should in-
clude at least the following:

- Converted source programs
- Flowcharts of the converted systems
- Listing of all job control language programs used
- Standard file labels
- File conversion parameters
- Operating instructions and technical notes
- Unit and system test reports

The following is a step-by-step summary of how a pro-
duction team should perform the translation of programs:

a. Materials are received by the production team and
processed by a Control Section. Each tape is analyzed
to ensure readability, and copied to backup tapes. Test
data are converted to target machine format. Standard
job control code is generated. Task estimates and a
schedule for the program translation are created by a
resource management system.

b. The source program is converted to the target language
by a multi-step process, using appropriate software tools. It has been found useful to first convert the source code to some intermediate language to permit standardized analysis and manipulation. This is particularly true if the conversion is to involve several source and target languages. The eventual restructured intermediate language program is then converted into properly formatted target language code. The target program listings and other documentation are collected and given to the project manager, who assigns the program to an analyst for completion of the documentation.

c. Corrections are made to the target program, again using appropriate software tools. The program is compiled until all diagnostics are resolved. Two programmers should then desk check every line to verify logical equivalence to the source program.

d. Testing begins with the aid of a cross-reference program, a tool to trap processing exceptions and allow continued processing, and a file compare program to verify output data equivalence. Unexecuted code is identified and carefully desk checked.

e. Unreferenced code is located and identified. Old data and procedure names are replaced where required by new names. The source code is formatted to installation standards to ensure the uniform appearance of all programs.

f. When program translation is completed, system enhancements are applied, and maintenance changes are identified and implemented.

g. The completed programs are returned to the project manager for a quality control check. The material is then processed by a control section and prepared for shipment to the implementation group. Backup copies of programs are stored on tape, along with microfilmed copies of listings for future reference.

Implementation

Implementation of the converted system should not take place until the systems software to be used has reached a satisfactory level of stability (this will be a subjective decision). The conversion manager should anticipate changes (upward) in the resources required to compile and execute the converted programs, and should allow for these changes in order to avoid serious degradation in throughput.

Unit and integration testing should be repeated on converted programs and test data once these are installed on the target system. Once this testing is satisfactorily completed, maintenance changes are applied and tested, problems are corrected, and retesting is performed. This process is repeated until all tests are successfully passed. It should be noted that this inclusion of maintenance changes entails the generation and conversion of production test data to be used in the testing process. Note also that no further changes must be applied to the production programs at this time.

In the meantime, the production database itself is converted and tested, and the operating system control language production stream is generated. Finally, production testing using the final version of programs, data, and control language programs takes place (acceptance testing), followed by an appropriate period of parallel testing.

Software tools

The description of the conversion phases included several references to software tools. A complete inventory of such tools is expensive, and this would be one of the factors which must be taken into consideration when contemplating an in-house conversion, since this inventory would not be of great value after the conversion.

There are several ways in which a software support inventory could be characterized, the simplest of which is according to the three principal conversion stages in which they are used:

a. Preparation: A file contents analyzer, a data extractor and modifier, and a data generator can all be used for creating test data. Utilities will also be required for creating backups of all programs, and for maintaining a current version of the software inventory to be converted and producing statistics (e.g., average program sizes) as required.

b. Production: The production stage will require software to perform source code to intermediate code translation, to analyze and restructure the intermediate code, to perform intermediate code to target code translation, and to translate test data files. Utilities will be needed to generate the operating system control stream and to apply code corrections to translated code. Additionally, software to produce cross-reference listings, to trap and identify exceptions, to identify unexecuted code, and to perform file comparisons will be needed for testing. A decompiler or depatcher will be needed if the conversion is from an assembler language to a higher level language.

c. Implementation: Software to validate the results of parallel testing, to identify and implement maintenance changes, and to convert the production data base is required.

Additionally, software aids will be useful in the management of the conversion project. As a minimum, software tools are needed for resource management (identify programs and categorize by source and content, estimate resource requirements, and monitor progress), and standards enforcement (format programs to installation standards, replace old names with standard names, etc.).

Managing the conversion project

A conversion project is only slightly different from any other software production project with respect to its management. Careful planning is required, the project must be initiated and, once initiated, it must be controlled. Finally,
there is a completion phase. If there is a significant difference between a conversion project management and a production project management it is one of emphasis; a conversion project requires (and allows for) more discipline and stricter adherence to procedures. If properly executed, a conversion is very much an assembly-line type of operation, where the total effort is broken down into well-defined tasks which are more dependent on experience and strict adherence to procedures than an innovation and ingenuity for their successful completion. This is true partly because of the high degree to which the conversion process can be automated. It should also be noted that many of the ground-rules for software production do not apply to conversion. For example, manpower and time are not generally interchangeable in a software production project, but, within bounds of good taste, they are in a conversion project.

Productivity rates also differ widely between conversion and development. Software development seems to proceed at some 12-20 lines per man-day for general application software, while conversion may proceed at rates as high as 400 lines of code per man-day.

A word of warning regarding project organization—a common mistake is to make the conversion staff a part-time group which participates in conversion activities but whose members continue to report to their parent organizations. This is an ingenious way to make a mess of the conversion, particularly since it will be nearly impossible to attribute responsibility for the mess to any one person.

The specific makeup and size of the conversion staff will vary with the conversion type and magnitude. The following membership, without specific quantities, is suggested for a large-scale project which will require contractor support. Following each staff category is an indication of the role of that category:

a. Project Leader
   (1) Planning
   (2) Project initiation
   (3) Project control
   (4) Project termination

b. Contracting—staff support advising on
   (1) Type of contract
   (2) Necessary clearances
   (3) Terms and conditions
   (4) Scheduling

c. Operations
   (1) Present status and future needs
   (2) Performance specifications
   (3) Scheduling
   (4) Inventory
   (5) Computer resources

d. Systems Programming
   (1) Inventory
   (2) Sizing a job
   (3) Performance specifications

e. Application systems developers
   (1) Inventory
   (2) Sizing
   (3) Performance specifications

f. Support programming—software tools for
   (1) Inventory
   (2) Quality control
   (3) Production
   (4) Testing

g. Material Control
   (1) Quality assurance
   (2) Backup inventory
   (3) Materials transmittal

h. Clerical—supports entire team

i. Analysis & Programming
   (1) Production
   (2) Testing
   (3) Implementation

Figures 1 and 2 suggest the organizational relationships among the staff components, and Appendix A lists the major tasks to be performed in the form of a checklist.

CONVERSION PROBLEMS

General

A conversion project results in a very large volume of material, and this results in control programs. Staffing required for the conversion project will not be needed at the culmination of conversion and will be diverted from ongoing development and maintenance. Furthermore, there will be periods of peak requirements. This will create serious management problems.

Machine time will also encounter periods of peak loads. This, unfortunately, conflicts with growing production due to the cutover of subsystems. Machine time availability takes on an inverse relationship to conversion requirements. In many cases, conversion requirements plus production requirements become more than the total of machine availability. This causes costly delays in the conversion schedule.

Low resources requirements estimates are caused by a lack of understanding of the conversion process. An esti-
A complete discussion of all the technical problems of conversion is beyond the scope of this paper. What follows is a brief discussion of some of the differences in machine architectures and file structures which can impact conversion.

Technical

Computer words vary in the number of characters that they contain causing problems in numerical accuracy and data movement. The accuracy program can be dealt with by increasing arithmetic precision, but this will increase storage requirements as well. The program in data movement is that programs can often refer to the storage of words or characters. When we translate these programs, the net effect is inconsistent treatment of data. A UNIVAC 1108 with a 36-bit word has six 6-bit characters. If we were converting to an IBM S/370, the 1108 program would move six characters in moving a word while the S/370 programs would move four characters in moving a word. To solve the program we have to determine whether or not the code refers to character data. This process often involves lengthy analysis, which requires personnel time, reduces automation, and increases time and costs of conversion.

Machine condition indicators are internal switches that are set for special conditions, e.g., overflow, zero divide, or invalid data. Problems arise when the source machine sets an indicator and the target machine either does not, or does so under slightly different conditions. For example, the 1401 would treat a space as a zero when doing any arithmetic. An IBM 370 will cause an interrupt called a data exception (OC7) when any non-numeric data appear as an operand in an arithmetic statement. This condition must be corrected in the translated program.

The format and the amount of information that is specified to define a file varies among languages. Some languages may...
require very little information to be specified, while other languages will require a great deal of information like record length, block size, or labels. Often there is even a wide variance between programs written in the same language because a language will allow, but not require certain specifications, or there will be different ways to define and specify file information in the same language, e.g., standard macros, user macros, or no macros (user-coded I/O).

The specifications of a file may be defined implicitly or explicitly. Explicit definitions are required in a COBOL program. An implicit definition might occur in an assembler program where a file is defined only by its use. Thus, the program and all references to the file must be analyzed to determine the appropriate file definitions. A similar problem also arises in data definition. A data item could be defined explicitly as a character field or not defined at all. All assemblers allow a programmer to specify arithmetic and/or binary operations on any data field. The actual use of a data item defines its implicit attributes. These may differ from their defined attributes in both data type and field length. When there is a discrepancy, an alternate definition must be generated to accommodate the circumstances. This condition is further complicated when implicit usages change because of word size variances.

No "standard" format exists for the recording of variable length records on tape or disk. Some of the ways in which the records are delineated are by delimiting records by special characters to indicate the stopping or starting point of each record, recording the variable length records as fixed length records by padding their length to a maximum consistent length, or inserting byte or word counts at the beginning of each record.

File organization becomes a consideration on any nonsequential file. Generally, indexed files contain pointers to indicate sequencing. Problems arise because variances are encountered in the location of the pointer, the format of the pointer, and the meaning of the pointer (some access methods will use actual track addresses, while others will use a relative record number within the file). The conversion of an indexed file includes getting the disk file onto tape, converting the pointer to the new format, and reloading the file to disk. The manner in which an indexed file is processed usually varies by language and machine. The differences may involve module communication or linkage to the access method for the system. Many direct access applications are developed using an algorithm to compute and determine the location of records on disk. For example, the disk addressing techniques of a source machine may be determined by an algorithm based on bytes per record, records per track, or tracks per cylinder. When converting to a relative addressing technique, or just a disk file with different characteristics, one or all of the above disk parameters could change.
The functions of OPEN and CLOSE in a program will vary. A tape-oriented program may OPEN and CLOSE each tape reel as a separate file; this coding must be eliminated when converting to a concatenated or disk file. Almost all I/O systems have different techniques for handling I/O errors. They also have different error recovery routines. Differences in these routines must be examined and handled. Communication with error routines must also be dealt with.

Data is frequently represented in different ways on the source and target machines. Often the differences can be easily dealt with, but representation differences can present significant problems. This for example, the BCD (Binary Coded Decimal) and EBCDIC (Extended Binary Coded Decimal) character sets have bit patterns for numbers (0-9) that have a greater binary value than the letters. Several other character sets do not maintain this same relationship and the bit pattern for the numbers has a lower value than for the letters. Conventions and requirements for representing (or not representing) signs on numeric fields often differ between machines produced by the same vendor and always differ between machines produced by different manufacturers. For example, on some machines a negative amount on character numeric fields is indicated by a "a negative zone" and all other numbers are represented by "no zone". On an IBM S/370 there are "-zones," "+zones," and "no zone" fields. On some systems a packed decimal number must always have a sign in the right four bits of the field. The Honeywell 8200 does not require a sign. If a sign is present, it can be one to four bits long and will appear in the left of the word, not necessarily adjacent to the field. Various Job Control Languages (JCL) are used to relate an application program to the hardware's operating system. Each is usually unique to a computer system. All functions that a program performs with the source operating system must be converted to functions to be performed by the target program and/or operating system. Differences among JCL may exist with regard to inter-module communication. Information (data) is often passed between subroutines within a running program. Sometimes, the data is left in a "common" area of memory by both the calling and the called subroutines. Communicated data must then be isolated and the address of the data passed between modules. Differences may also exist in the roles of the application program and the operating system. In the past, with programs running sequentially in computers, the operating system allowed the program to determine its own hardware usage, file allocation, core allocation, and time requirements. When multiprogramming, these functions are removed from the application programs and supplied by the operating system.

In addition to system differences, the differences created by individual organizational programming practices can create significant problems in a conversion effort. In fact, these represent the major factor impacting cost. Sometimes installations using low level languages do not have stringent programming standards. This often results in non-uniform, inconsistent programs. COBOL or PL/I forces a certain degree of standardization. The transformation process must be designed to handle each of the different circumstances and to produce code in a standard high level language. In-sufficient, out-of-date or non-existent documentation is a significant problem for most conversions. The conversion process does not require extensive documentation since we use the source program as the specification for the target program, but system level documentation (or job/file flow) is needed to restructure a system and record layouts are required for file conversions. Systems are often designed with specific hardware/software limitations or user options in mind. Examples are the use of checkpoint/restart for a program that runs too long, using overlays because of core limitations, using a disk file instead of a tape file because of a shortage of tape drives, or running a sort separately instead of as an exit in our application program. Many of these conditions can be changed to improve efficiency when the program or system is converted.

Appendices B and C outline some specific sources of conversion difficulties which may occur in COBOL and FORTRAN programs, respectively. Reference 4 gives an excellent survey of transferability problems of programs written in these and other languages, and Reference 5 suggests techniques to be used in minimizing FORTRAN conversion problems.

CONTRACTING FOR CONVERSION SERVICES

Success in conversion is in large measure dependent on experience and few if any data processing organizations possess this experience. There is little reason why they should—conversion is not an ongoing enterprise. The experience required is not of the "we have three professionals who have gone through a couple of conversions" type. Rather, experience, to be useful, must involve an entire team who has performed enough conversions to become proficient in the techniques and tools to be used, and predictable in its productivity and performance quality. Such experience is most apt to be found in software services or conversion services contractors. Generally, it would be wise for organizations to avail themselves of contractor support for conversion. This is true particularly since, contrary to popular opinion, the software producers are not the best qualified people to convert the same software. If anything, they are the worst qualified, since they probably cannot resist the temptation to "improve" the system while converting it.

Contracting methods vary greatly. The best, most sound, contracting is probably done by the U.S. Government under its Federal Procurement Regulations. Useful guidelines can also be obtained from Reference 7. The following are offered as general factors to be considered.

The first step in a procurement is the same as the first step in the conversion planning, namely, a requirements analysis. Once this is done, the contracting personnel can begin to plan for the type of contract which may be required by the specifications, the time required to prepare a request for proposals (RFP) to potential contractors, the time required for negotiations and evaluation of proposals, and the time required to make an award. The availability of funds must also be determined at this time. Our experiences in-
dicate that the procurement may take from 22 to 38 weeks to complete.

The technical staff can ease the burden of the contracting staff by preparing a complete procurement request (PR). This will include a thorough statement of the scope of work (supplies and services, etc.), reporting requirements, property or facilities to be provided to the contractor, and a list of prospective sources of support. Upon receipt of the PR the contracting officer can assemble his procurement staff which, for large or complex procurements may include negotiators, a cost/price analyst, an inspector, legal counsel, and auditors.

A complete discussion of contractor evaluation methods is beyond the scope of this paper, but the principal ones are listed below:

- Cost only—pick low bidder among qualified ones. If the specifications are adequate this is not as bad a method as it is reputed to be.
- Cost plus "desirables"—the basic bid price is modified up or down according to the quality or number of "non-mandatory" items a contractor purposes. This is a superficial way of giving "bonus points" which has little to recommend it in this author's opinion.
- Cost evaluation plus technical evaluation—it is common to attempt to combine "cost points" with "technical points" to determine a winner. This is usually motivated by a desire to substitute subjective judgment for a sound evaluation. One legitimate way of incorporating a technical evaluation into the overall evaluation scheme is to use the results of a technical review to determine a "technical competitive range," to eliminate all bids falling outside this range, and to make the award to the lowest of the remaining bids.

ESTIMATING CONVERSION COSTS

Conversion costs for an in-house conversion (i.e., performed entirely by a staff of programmers from the organization undergoing the conversion) cannot be accurately estimated. The difficulties of estimating production efforts are well-known. Yet, software production is something which programmers do all the time. They do not do conversion all the time. Furthermore, the in-house staff does not have the tools and procedures required, and even if it acquires them it has little or no previous experience in using them.

One can determine reasonable estimates of conversion costs if a contractor is used for the production stage, which is the most costly of the various conversion stages. The various tasks associated with planning and implementation can be itemized, resources required for each of these are estimated, and this results in a cost estimate for these portions of the conversion.

Estimates for the production stage can be derived by reviewing past conversion efforts and prices bid on these efforts. The Federal COBOL Compiler Testing Service has compiled a substantial data base of cost and productivity figures. A complete description of the resulting cost model would require an entire paper itself, and will therefore not be presented here. A few figures can however given general indication of potential costs.

- The cost of converting a line of COBOL code (to COBOL) may range from $.40 to $6.00, with $.40-2.00 being a most likely band, and $.65-1.00 being a good average figure for reasonably clean COBOL programs requiring no extensive file restructuring on additional documentation. FORTRAN to FORTRAN conversion costs are similar to COBOL-COBOL costs.
- Data on PL/I to PL/I conversion is scarce, but indications are that such a conversion would be 10-20 percent costlier than a COBOL-COBOL conversion. Assembler code—COBOL/FORTRAN will cost from $2+ to $8 per source line.
- Productivity figures range from 300-500 lines per man-day in FORTRAN-FORTRAN or COBOL-COBOL down to 20-100 lines per man-day in an assembler-COBOL/FORTRAN conversion.
- Extensive documentation (user guides, narratives, etc.) can cost up to 40 percent of the total per line cost. In-house costs for planning, preparation, and implementation can be up to 50 percent of the total costs.

There are also several qualitative observations which can be made from an analysis of a sizable conversion data base:

- Conversion costs are more dependent on the source machine than on the target machine.
- Knowledge of the application is not critical in performing the conversion; but information regarding the application must be available as required. (Redesign is of course excepted here.)
- Known complexity will not unduly influence costs, but complexity which comes as a surprise can cause havoc.
- None of the above holds for real-time systems.
- Conversion of data base systems is not well understood.
- Conversion estimates should not be attempted by people who do not have access to a sizable data base of information and who do not do this on a regular basis.
- Cost models consisting of a handful of formulas based on a handful of parameters are worse than useless; they are dangerous because they give the impression of authority.

CONCLUSION

Conversion is a disruptive, largely non-productive process which must nevertheless be faced at some point in the life of a data processing organization. This paper has presented some ideas which might be of assistance to someone contemplating a conversion, the most important of which are
that conversion requires careful planning, and that it should not be attempted without the services of professionals.

REFERENCES


APPENDIX A—SYSTEM CONVERSION CHECKLIST

PLANNING
- Collect inventory and define scope of work
- Analyze differences between source and target systems
- Develop conversion plan and schedule
- Estimate resource requirements
- Assign conversion responsibilities

PREPARATION
- Develop or acquire software tools required
- Collect and package programs
- Prepare adequate test data
- Create test output using test data
- Collect related materials

PROJECT CONTROL
- Standardize conversion procedures
- Establish reporting requirements
- Monitor project status
- Assess quality of converted programs

PRODUCTION
- Convert source code
- Code corrections required
- Prepare test job control
- Format programs to standard
- Convert test files

TESTING
- Conduct unit test
- Conduct system test
- Conduct parallel test
- Ensure accuracy of converted programs
- Ensure test data adequacy

INSTALLATION
- Implement maintenance changes
- Prepare production job control

Convert production files
Cutover to production

APPENDIX B—COBOL PROGRAMMING PROBLEM SOURCES

GENERAL:
- Requires operator intervention in processing
- Requires operator console input(s)
- Has checkpoint/restart capabilities
- Contains an interface to database system
- Requires object code patches not incorporated into the current source
- CALLS to an assembler or other non-COBOL language subprogram
- Uses overlays or segmentation

IDENTIFICATION DIVISION ENTRIES:
- Entries out of order with respect to the ANS COBOL Standard

ENVIRONMENT DIVISION Entries

FILE CONTROL
- I-O-CONTROL
- DATA DIVISION Entries

RECORDING MODE
- BLOCK CONTAINS 0
- USAGE IS COMP-1 short precision floating-point data
- COMP-2 long precision floating-point data
- COMP-3 internal decimal data (packed data)
- COMP-4 binary data

WORKING-_STORAGE SECTION
- Logic of the program expects certain initial values when data has not been initialized
- REdefines
- Occurs depending on
- Bit level data fields (non character aligned)
- Logical switches
- Logical masks
- Floating point literals
- Floating point fields
- Signed zero
- Unsigned numeric fields used in computations

INDEX
- Subscripts
- Sort description SD-names

LINKAGE SECTION
- Linkage entries

COMMUNICATION SECTION
- Communication description CD-names

REPORT SECTION
- Report writer description RD-names

PROCEDURE DIVISION ENTRIES
- Program logic is sensitive to numeric precision
- Program logic dependent on the collating sequence, i.e., blank
- Logic dependent on HIGH VALUE or LOW VALUE
- Logic dependent on rounding or truncation of numeric results
Logical shifts or bit manipulating
ALTER
CLOSE
COMPUTE
EXAMINE
GO TO DEPENDING
Certain MOVE statements

OPEN
PERFORM (but not PERFORM . . . THRU . . .)
SEARCH
COPY
TRANSFORM
WRITE
SORT
LABEL