The Effect of Language on Software Costs

Susan H. Johnston
Robert Morris College
Coraopolis, Pennsylvania

Abstract

The choice of programming language for software development can have a great impact on costs for both development and maintenance. In the recessionary climate, with the emphasis on cost efficiency, assessing the effect of the various features of a programming language on cost, efficiency and productivity seems prudent. The entrenched traditional third generation programming language, COBOL, is compared and contrasted with MUMPS, regarding the structure of programs, operating environments, language extensions, data types, sorting, syntax and functions. The flexibility of MUMPS in these areas is shown. MUMPS' superior performance in a benchmark test conducted for creating a database is explored. MUMPS' lack of requirements for language extensions in accessing a database, but ability to accept extensions such as SQL, are contrasted with COBOL's requirements for such language extensions. MUMPS' language features have a positive effect on productivity, efficiency, and cost reduction which can ultimately affect a company's competitive position in the global marketplace.

Introduction

When reading most current computer literature, one is struck by the trends toward downsizing, rightsizing and outsourcing. One corporate official predicted that at least fifty percent of their mainframe applications will be rewritten over the next five to ten years.[3] In the recessionary economy companies are attempting to reduce both hardware costs and software maintenance costs. Chief information officers are questioning the way that data processing has traditionally been done. Software development and maintenance costs are a product of the programming language used for development, in addition to management attitudes toward development and maintenance. What role do programming languages play in this upheaval in the data processing industry?

In speaking of programming languages, the general public has heard about FORTRAN, COBOL, C, and PASCAL, among others, but many people have not heard of the language MUMPS, also called "M". Considering that M was the third language standardized by the American National Standards Institute (ANSI) after FORTRAN and COBOL, it is surprising that it is not better known.[2] COBOL (Common Business Oriented Language) has traditionally been well known for its widespread use in business applications. M has been thought of as a medically-oriented language since it was originally developed in a hospital setting for use in the medical field, but M is widely used in other business applications such as inventory control, accounting, and manufacturing. M has been used to develop rule-based expert systems.[17] M has also been promoted as a rapid prototyping tool.

Productivity Comparisons

In any computerized system data are stored, retrieved and, depending upon the application, manipulated to some degree. Methods of accessing and manipulating data vary widely in their efficiency due to the manner or structure in which the data are stored. Productivity and efficiency in the processing of data are considered of primary importance in computer systems.

In a benchmark study done by a department store in Spain when accepting bids for its computer system, M was one of the competing languages and database systems. Measurements were taken for disk occupation, processing speed and number of lines of code required. The languages compared were MUMPS, RPG III, COBOL, BASIC and FORTRAN. During the test a database had to be created from scratch and a long and short query run. "On the Database Creation test, MUMPS was from 30 to 7 times faster than the conventional systems. The MUMPS database needed only between 1/4 and 1/2 of the disk space used by the competitor's databases."[1] M compared very favorably in the

number of lines of code required because it needed less than one third of the lines of code than the other systems in the study. MUMPS was tested against its competitors on both 32-bit and 16-bit computers and was beaten on the 32-bit computer only once.

The test between MUMPS and COBOL was run on the same computer hardware, so that just language differences could be taken into account. In database creation MUMPS took thirteen minutes while COBOL took 2 hours. For the long database query MUMPS took 1 hour 37 minutes versus COBOL's 2 hours 35 minutes. MUMPS' database system occupied 5.4 megabytes of disk space versus COBOL's database system which occupied 21.7 megabytes, over four times more than the MUMPS database system. MUMPS only required 183 lines of code versus COBOL's 1050 lines.[1]

	MUMPS	COBOL
Database creation	13 minutes	2 hours
Disk space occupation	5.4 meg	21.7 meg
Lines of code	183	1050

Number of lines of source code has been one measurement used in software metrics as a correlation for initial programming time and maintenance time.[18] "Program sizes in COBOL are typically 20 to 30 times the size of an equivalent MUMPS program. This has an obvious effect on disk storage requirements, but an even larger impact on memory requirements, computer power and execution time."[7]

More recently Digital Equipment Corporation, using Digital Standard MUMPS (DSM), "received the highest rating in a benchmark testing transaction processing performance."[6] The Transaction Processing Performance Council (TPC) Benchmark A test is an industry standard test which measures the number of transactions per second and the cost per transaction per second. In order to qualify for a Benchmark A test, strict compliance with rigorous conditions must be kept and presented to the TPC.[8] Digital's participation in the TPC-A test in 1990 "established DSM as the top achiever in transaction processing speed and cost efficiency."[6]

What makes M quicker in database creation and access while occupying less disk space and using less lines of code? What features does M possess that COBOL does not, or vice versa, that make M more cost effective?

Contrast the Features of COBOL and M

Different programming languages vary as to the structure of the programs, operating environments, language extensions, data types, sorting, syntax and functions. Since M and COBOL were among the first languages to meet ANSI standards they will be compared in each of these areas, in an attempt to assess what made M more cost efficient and productive in the above tests. Positive and negative aspects of each language will be contrasted.

Program Structure

Older COBOL-74 programs require the declaration of four divisions: Identification, Environment, Data and Procedure. The Identification Division documents the program. The Environment Division, which consists of the Configuration Section and the Input-Output Section, links the COBOL program to the system environment in which the program will be run. The Configuration Section identifies the computers used to compile the source program and run the object program. The Input-Output Section describes all of the logical data files and associates them with physical files through an input/output device.

The Data Division contains two sections, the File Section and the Working Storage Section, which are used to identify logical files and data to be used within the program. The File Section contains file description entries for each input/output file and sort-merge-file used in the program. Record description entries describe logical records within files. More than one record description entry can be specified for the same storage area. The Working Storage Section declares other fields necessary in the processing of the program such as total fields, status condition fields and additional file and record specifications. The Data Division may also contain a Linkage Section which identifies data passed from another program. The Procedure Division contains the program logic using various reserved words in COBOL statements. There are over 300 reserved words in COBOL which cannot be used as user-defined variable names within a COBOL program and must be spelled out in full in order to use them. This makes COBOL a lengthy, wordy language, but the English-like nature of the language makes the program logic fairly easy to read and somewhat self-documenting. (See example following for divisions of COBOL program. See Exhibit A for a complete COBOL program.)

```
IDENTIFICATION DIVISION.
PROGRAM-ID.     SAMPLE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION
INPUT-OUTPUT SECTION.
FILE-CONTROL.
  SELECT SAMPLE-FILE
       ASSIGN TO INFILE.
DATA-DIVISION.
FILE SECTION
FD  SAMPLE-FILE
01  SAMPLE-RECORD
WORKING-STORAGE SECTION.
PROCEDURE DIVISION.
```

The newer COBOL-85 standard makes optional many of the prior requirements. The Author, Installation, Date-Written, Date-Compiled and Security paragraphs of the Identification Divisions are now obsolete. The Environment Division and the Configuration Section are now optional to enable the use of nested programs and subprograms. The Source-Computer and Object-Computer paragraphs of the Configuration Section are also optional. User-defined words can be identical to system-names in COBOL-85 because the COBOL-85 compiler is able to differentiate between a user-defined word and a system-name based on the context in which it is used.

The Data Division is also optional, in addition to the Label Records and Data Records clauses in order to allow for nested programs and subprograms. The nested subprograms could optionally contain the Environment and Data Divisions as programming circumstances dictate, or they could access data in the main program that is declared as global. The record clause can now be coded RECORD IS VARYING IN SIZE FROM nn TO nnn CHARACTERS DEPENDING ON user-defined variable, to allow for variable length records, where n stands for an integer. The reserved word Filler is optional also. The Procedure Division is optional in the COBOL-85 standard in order to permit prior compilation of the first three divisions for correct syntax.

M programs do not require the declaration of any divisions within a program. An M variable can be defined, used and deleted within the same line. M is designed to run on various types of hardware, from mainframes to microcomputers and personal computers. It can be a standalone operating system or run under various operating systems such as VM, VMS, UNIX, AOS, DOS and OS/2.[2] The type of system is never declared within an M program. Source and object computers do not have to be declared.

M commands can be abbreviated to one letter (i.e. SET - S, HANG - H) and functions can be abbreviated to two or three characters (i.e. ORDER function - $O, LENGTH function - $L, TRANSLATE function - $TR). In general, there are no reserved words. For example, the letter S is not restricted by M to indicate the SET command, but can also be used as a variable. M interprets the context in which a letter is used, as to whether it is interpreted as a command or a variable name. M does not have any restrictions as to variable names, because the restriction of reserved words does not apply.

Operating Environment

COBOL requires a command language which constructs access methods for files. In the IBM environment the command language depends upon the operating system. JCL (job control language) is used under IBM's OS/MVS to specify every file used in a COBOL program. IBM's VM/CMS operating system uses the command language CMS (Conversational Monitor System) which is designed for interactive processing. In the IBM environment an additional language to COBOL is needed just to process file data. COBOL can be made interactive to the user through additional language extensions such as CICS, but the COBOL-85 standard does not provide for interactive programs. "Computer hardware requirements are much higher for COBOL than MUMPS."[7] This statement is true because COBOL usually runs on large mainframes or microcomputers with substantial added RAM. COBOL is a compiled language because it compiles it source code into object code which is stored and executed.

M's unique syntax includes a job control language, linkage editor and database management system, and a data communications monitor, in addition to an application programming language.[13] It does not need additional languages to access the data in its global variables (equivalent to files), because the access method is built into the language. Standard M programs are portable from one environment to another, because they do not need the support of additional languages and operating systems.[12] Individual fields within records can be accessed without explicitly stating the file structure, including fields and field lengths. The field lengths within a record may be of variable length without affecting M's accessing method. M is usually run on minicomputers or microcomputer networks in a multi-user environment for

interactive database applications. M is an interpreted or a hybrid compiler-interpreter system.

Language Extensions

Without any language extensions for accessing database management systems, such as IMS, CODASYL, IDMS, Btrieve and SQL, COBOL is capable of accessing sequential files, indexed sequential files and hash files. There is no data independence using traditional COBOL programming due to the explicit file and record declarations necessary within programs.[4] In order for COBOL to access a database structure, additional code embedded into COBOL is necessary. This necessity for embedding code, such as SQL, IMS, CODASYL or DB2, adds the capability for accessing related database records, but adds additional code to an already wordy language. The various language extensions to COBOL make it capable of accessing various types of databases, such as hierarchical, network and relational.

One of M's great strengths is that a database management system, including storage and retrieval facilities, is an integral part of M. M uses subscripted global and local variables, which are built into the language, instead of external input/output files to store and retrieve data. M's permanent storage of data in its database system is accomplished through global sparse arrays, which use a hierarchical tree structure.[2] Sparse arrays promote efficient utilization of memory space through allocating space only for elements of the array with data. Prior allocation of space is unnecessary because space is allocated as needed for data storage. Global variables are shared by various users and persist after a program or interactive session has finished. Both local and global variables, including fields, do not have to be declared explicitly in a division at the beginning of a program. They can be created, used and deleted (KILLED) in the same line of M code. Globals are not opened and closed and can be accessed sequentially or randomly. Subscripts of global or local arrays can be numeric or string values.

Language extensions to M have been made to allow M and other systems to share data. SQL and M have been combined to allow M access to data stored in a relational structure. The combination of the two languages has increased productivity and efficiency, in addition to allowing access to both hierarchical and relational databases. An 80% increase in storage efficiency was gained in one application converted from COBOL to M combined with SQL.[5] Additionally, an interface between FOCUS and Digital Standard MUMPS (DSM) allows M applications to access data from other systems such as ADABAS, DBMS, RMS, Rdb, Ingres, Oracle, Sharebase, Sybase and Teradata using SQL.[9] These extensions provide access to M's hierarchically stored data along with data stored in other formats, such as relational, optimizing data sharing and access. These language extensions to M should mute the criticism that M is not compatible with other systems.[12] The Open MUMPS Interconnect which provides for a standard protocol for accessing databases across a network is currently a Type A extension.

Data Types

COBOL data is declared as numeric, alphanumeric or alphabetic in the Data Division of the program using PICTURE clauses. Each record must be explicitly defined as containing specific fields with definite lengths. The REDEFINES clause is used in COBOL to assign additional data names or PICTURE clauses (data types) to a specific area of storage. In this way a field that is defined as alphanumeric can be redefined as numeric and calculations can be performed on the data in the field. COBOL-85 permits the redefined field to be longer than the field being redefined. COBOL-85 also permits the PICTURE character string to be continued over two lines of source code. In COBOL-85 the editing symbols of a comma or decimal point may be the rightmost character in a PICTURE clause as long as the period indicating the end of the sentence immediately follows the rightmost decimal point or comma.

```
    05 INVOICE-NO              PIC  9(8).
    05 INVOICE-NO-X  REDEFINES INVOICE-NO.
       10    INV-NO-X          PIC  9(7).
       10    FILLER            PIC  X.
    05 RETURN-NO REDEFINES INVOICE-NO
                               PIC X(8).
```

In M, all data are typed as character strings with no explicit declarations of specific types of data necessary. The context of the operation defines whether a particular field is interpreted as a string, a numeric value or as a truth value. Placing an arithmetic operator (addition "+", subtraction "-", multiplication "*", exponentiation "**", division "/", integer division "\", and modulo "#") before a variable causes it to be interpreted as numeric. The expression +"-1.25" would be interpreted as -1.25 and +"ABC" would be interpreted as 0,

thus calculations can be performed on any expression. M has a function $FNUMBER which will format a number for display with signs, decimal places and parentheses as requested within the function. A display record with explicit fixed length field declarations is not necessary in M. String operators and functions (contains "[", follows "]", equals "=", concatenate "_", $LENGTH, $EXTRACT, $FIND, $ASCII, $CHAR and $PIECE) will interpret any variable as a string.

Sorting

Sorting files for the manipulation of data for a report or on-line presentation consumes a significant amount of system resources in conventional computer systems. M's sort requires no time consuming file declarations, making it quick to use. COBOL follows the more conventional sorting methods.

The traditional sort within a COBOL program involves setting up a sort definition for the sort file, including explicit field definitions, in the File Section of the Data Division. The sort must be specified in the Procedure Division by identifying the input and output files, the key to be sorted on and whether the sort will be ascending or descending. COBOL-85 eliminates the requirement for specifying a section name for an INPUT or OUTPUT PROCEDURE on SORT and MERGE statements. The requirement is for a procedure name which can be either a paragraph name or a section name. A new clause WITH DUPLICATES IN ORDER has been added that will return records with duplicate sort keys, in their original order in the file, first in, first out. The sort file definition can contain the clause, IS VARYING IN SIZE FROM nn TO nnn CHARACTERS DEPENDING on user-defined variable, to allow for variable length sort records (n stands for integer).

M's collating sequence starts with null, followed by canonic numbers then all other strings. M's globals automatically store information in this collating sequence.[19] To retrieve information in ascending or descending order is simply a matter of using the ORDER function ($O). No additional file and record definitions have to be explicitly declared. To retrieve information from a global in the order of a different key, an index can be created (such as name, social security number) within a program and the data will be retrieved by the ORDER function, using the new index, in name order instead of social security number order. The index can be deleted immediately after use, so the services of a database administrator to create a different key are not necessary.

Syntax Comparisons

Syntax variations between languages contribute to the ease of programming, productivity and efficiency of the various languages. Differences between M and the COBOL-85 standards are discussed below.

COBOL's MOVE command is equivalent to M's set command. The command in COBOL would be: MOVE X to Y. In M the command would be: SET Y=X. COBOL-85 now permits a numeric edited field to be moved to a numeric field or another numeric edited field. In order to permit acceptance of numeric data from a terminal, the MOVE command de-edits the numeric-edited field prior to moving the data. All of the editing characters such as commas, decimal points, dollar signs and positive or negative signs would be removed from the data and the remaining numeric data values moved to the unedited field. M can set any variable equal to any value and has no limitations on movements between fields. M's variable length fields do not restrict movements between fields, because all fields can be of varying length strings.

COBOL's PERFORM is similar to M's DO command because both commands are used to transfer control to labeled sections of code. Control is returned in both cases to the statement following the DO or the PERFORM. The new in-line PERFORM statement of COBOL-85, which eliminates transfers of control to performed procedures, compares with M's argumentless DO command with block structuring. Instead of specifying a procedure name, the statements coded in line after the PERFORM or the DO will be performed. An END-PERFORM terminates the PERFORM statement and a QUIT terminates the DO statement.

```
PERFORM
    (statements to be performed)
END-PERFORM.

DO
.   statement to be performed
.   statement to be performed
QUIT
```

COBOL's PERFORM/UNTIL has been modified in COBOL-85 to allow optional specification of WITH TEST AFTER or WITH TEST BEFORE until a specified condition has been met. This change was made to change the minimum number of iterations, if WITH TEST is coded, from 0 to 1. M's FOR/DO command can specify a specific number of

iterations. This command can include a QUIT with a post-conditional, which will terminate the loop if a certain condition is met. M also has the argumentless DO command which uses the QUIT with post conditional to terminate the loop, thus the number of iterations does not have to be specified. M's repeat/until would be coded FOR DO label or block structured code Q:postconditional. M's do/while would be coded FOR QUIT:postconditional DO label or block structured code. This would correspond with COBOL's WITH TEST BEFORE or WITH TEST AFTER.

Explicit scope terminators in COBOL-85 such as End-Add or End-Write make the range of each statement more evident to the reader, and promote structured coding. M has the QUIT command which generically ends routines, functions and FOR loops, but it does not specifically indicate which function or routine is being ended. COBOL-85's explicit scope terminators specifically identify which function is being terminated which helps in following the program logic.

```
END-ADD          END-READ
END-CALL         END-RETURN
END-COMPUTE      END-REWRITE
END-DELETE       END-SEARCH
END-DIVIDE       END-START
END-EVALUATE     END-STRING
END-IF           END-SUBTRACT
END-MULTIPLY     END-UNSTRING
END-PERFORM      END-WRITE

QUIT
```

COBOL's CALL program command is equivalent to M's DO ^label command, where the ^label means an external program name. Each language is calling another program to be performed. In COBOL a Linkage Section must be declared in the Data Division to identify fields of data from the called subprogram. In COBOL the syntax for the calling program is CALL "PROGRAM-NAME" USING DATA1, DATA2, etc. The called program must have the Procedure Division coded as PROCEDURE DIVISION USING DATA1, DATA2, etc. The parameters are matched using the position of the parameters.

In M the syntax in the calling program, which contains the actual parameter list, is DO PROGRAM(Data1,Data2). The called program, which contains the formal parameter list, is coded PROGRAM(Data1,Data2). The actual list of parameters can be shorter than the formal list of parameters and the formal parameters are initialized (by an implied NEW command) before the subroutine or called program is executed. In M the parameters are also matched using position. Control is returned to the original program in both cases after the CALL or DO is completed. M uses the DO command for performing a subroutine within a program (DO label) or for performing an external routine (DO ^routine). The only difference in the syntax is that the external routine requires an up-arrow or caret (^). No Linkage Section is necessary in M.

The EVALUATE statement in COBOL-85 was developed to replace several IF statements for multiple condition testing and implements the case structure in COBOL. It is comparable to the SELECT statement in M. The EVALUATE statement checks a variable for a certain value and executes a different function depending upon what value is contained in the variable. The $SELECT function ($S) evaluates an expression as to whether it is true or false and, if it finds a true expression, it executes the value corresponding to the true expression. The $SELECT function can be coded with an else value just in case no true expression is found. As can be seen below, the COBOL Evaluate statement is much more wordy than M's $SELECT function, but it is very readable. The $SELECT function is more concise. An M programmer would find it readable.

```
EVALUATE STATEMENT - COBOL

    Evaluate TYPE
        When 1
            Move 'a' to NAME-CODE
        When 2
            Move 'b' to NAME-CODE
        When 3
            Move 'c' to NAME-CODE
        When other
            Move ' ' to NAME-CODE
    End-Evaluate.


SELECT STATEMENT - M

$S(T=1:N='a',T=2:N='b',T=3:N='c',1:N=' ')
```

COBOL-85 had been updated to obtain the day of the week through the use of ACCEPT identifier FROM DAY-OF-WEEK statement. The field used for the identifier field must be a one-digit unsigned numeric field. Days of the week are represented numerically from 1 to 7 for Monday through Sunday. In M the $HOROLOG function, $H, will obtain the

date and time. To obtain the day of the week there are common utilities available in the various systems.

The IF statement was updated in COBOL-85 to include the reserved word THEN, making the syntax IF-THEN-ELSE, which is more structured wordage instead of just IF-ELSE. M uses the IF-ELSE syntax. In M the IF syntax affects the rest of that command line only. The ELSE syntax also affects only the rest of that command line. The IF command sets the value of a system variable $TEST to either true (1) or false (0) based on the conditions contained in the IF statement. The ELSE statement is executed only if the value in $TEST is false (0). The $TEST system variable is also set by other commands such as a LOCK or READ with a time-out. This could potentially cause the ELSE statement to execute or not execute in error.

COBOL-85 has been revised to include two new relation operators not previously included in COBOL-74, the GREATER THAN OR EQUAL TO, >=, and the LESS THAN OR EQUAL TO, <=. In M, greater than or equal to is coded as NOT LESS THAN, '<. Less than or equal to is coded as NOT GREATER THAN, '>.

COBOL-85 has two new class conditions, ALPHABETIC-UPPER and ALPHABETIC-LOWER to enable testing for upper and lower case. These two new classes are in addition to the ALPHABETIC class condition, which tests for both upper and lower cases including blank spaces and the NUMERIC class which tests for numeric. This compares to M's pattern match operator ?U for upper case or ?L for lower case and ?A for alphabetic. ?A, ?U and ?L do not include blank spaces. M's pattern match operator allows for checking for ASCII control characters (?C), the entire ASCII character set (?E), numeric digits (?N) and punctuation characters (?P).

M can check for specific numbers of these characters in addition to checking for string literals such as DATA?1"z", to check for the letter "z" in the variable DATA. M can check a variable to see if a certain series of characters is contained within the variable. The syntax to check for the two formats of zip code would be coded DATA?5N!(DATA?5N1"-"4N). Thus M is capable of checking a variable for very complex patterns of characters including ranges of characters. The pattern match operator is evaluated to either true or false.

COBOL-85 has added a class facility which is user defined in the SPECIAL-NAMES paragraph of the CONFIGURATION SECTION in the ENVIRONMENT DIVISION. This user-defined class facility can be tested with a class-condition IF statement. M can set a variable equal to the values necessary to define the class and the CONTAINS clause can be used to see if data is contained in the values assigned to the variable. The ALPHABET clause in the SPECIAL-NAMES paragraph allows the choice of three collating sequences. STANDARD-1 is the ASCII code (American National Standard Code X3.4-1977 for Information Interchange). STANDARD-2 is for the International Standard 646 code for Information Processing Interchange. NATIVE is for the native code for the computer, which is EBCDIC (Extended Binary Code Decimal Interchange Code) for most IBM systems.

COBOL-85 has added condition expressions that represent a false path to conditions that in COBOL-74 could only be tested for true conditions. READ AT END can now be coded for a false condition READ NOT AT END or READ INVALID KEY can be coded for the false condition READ NOT INVALID KEY. M can code a negative for a conditional using the not (') logical operator.

```
READ MASTER-FILE
    KEY IS MASTER-KEY
INVALID KEY
    PERFORM ERROR-ROUTINE
NOT INVALID KEY
    PERFORM ROUTINE-1
    PERFORM ROUTINE-2
END-READ.

IF '$DATA(^MASTER(MKEY)) D ERROR Q
```

COBOL-85's USAGE IS BINARY clause replaces COBOL-74's COMP clause to specify binary representation in computer storage. Also, USAGE IS PACKED-DECIMAL in COBOL-85 replaces the nonstandard COMP-3 of COBOL-74 in order to provide for packed decimal representation. No equivalent clause for storage is necessary in M.

The COBOL-85 standard also added the INITIALIZE statement in order to initialize elementary fields within a group. This statement helps to reduce the coding involved in initializing group fields. Alphanumeric and alphabetic fields, including edited fields, are initialized to spaces and numeric fields, including edited fields, are initialized to zero. There is also a REPLACING BY clause to provide the means to initialize to other values than zeros or spaces. In M, groups of fields can be initialized using the SET statement, i.e., SET (A,B,C)=0, sets fields A, B and C to zero. The SET statement can also be used to initialize fields to other values.

No additional or new statement was needed in M to provide for this initialization capability.

COBOL's original INSPECT statement converts individual characters in a field from one value to another. The new INSPECT/CONVERTING statement permits the replacement of multiple characters in a field. The characters are matched positionally. The syntax INSPECT DATA-FIELD CONVERTING "XYZ" to "ABC" would replace X with A, Y with B and Z with C.[11,21] M's TRANSLATE function ($TR) with the syntax $TR(VARIABLE,REPLACE,WITH) or $TR(VARIABLE, REPLACE) substitutes characters in REPLACE that are contained in VARIABLE with characters from the WITH string. It is a character by character substitution.[11]

Using the concept of reference modification, it is possible in COBOL-85 "to reference a portion of a data field without using REDEFINES or group items in the data description."[15] Assuming two data fields, X defined as PIC 9(10), representing a telephone number including the area code, and Y defined as PIC 9(7), the syntax using reference modification would be: MOVE X(4:7) TO Y. The 4 is interpreted as the starting byte and the 7 is the number of bytes being moved. The data in A would be moved starting with the fourth byte and moving seven bytes. Thus Y would contain the phone number minus the area code. The second item which specifies the number of bytes being moved is optional. If not included, the move will be from the starting byte specified to the end of the field. The new reference modification in COBOL-85 compares to the EXTRACT ($E) function in M. It extracts "a substring from a target string by character position(s)." Thus $E("COBOL-85",5) would return an "L" or $E("ANSI STANDARD MUMPS",6,99) would return "STANDARD MUMPS". To return the first character of an expression the syntax would be $E("VARIABLE") which means $E("VARIABLE",1), without coding the '1'.

COBOL's STRING statement joins (concatenates) multiple sending fields into one field. The DELIMITED BY clause specifies which character marks the end of the sending field. To use all of the characters in the sending field SIZE is specified as the delimiter, otherwise only the characters to the left of the DELIMITER are used in the STRING operation.[14] The POINTER option specifies the starting position for characters to be placed in the receiving field.

```
STRING FIELD-A, FIELD-B, FIELD-C
    DELIMITED BY '^'
      INTO TARGET-FIELD
      WITH POINTER POSITION.
```

M's concatenate operator, which is the underscore, "_", attaches one string to the end of another string. If A="M ", B="vs. " and C="COBOL", the syntax: WRITE A_B_C would result in "M vs. COBOL".

COBOL's UNSTRING statement separates one field into one or multiple receiving fields. The DELIMITED BY clause indicates what character separates the fields in the sending field. More than one delimiting character can be specified using the OR clause. The delimiter will also be placed in the receiving field if the DELIMITER IN clause is coded. The number of receiving fields which are used can be totaled with the TALLYING IN clause and the COUNT IN clause counts the number of characters in a receiving field.[14]

```
UNSTRING INITIAL-FIELD
    DELIMITED BY '^'
    INTO FIELD-A
      COUNT IN COUNT-A
        FIELD-B
      COUNT IN COUNT-B
    TALLYING IN TALLY-COUNT.
```

The UNSTRING function is similar to M's $EXTRACT function as explained in the above paragraph, which uses character position to select a substring from the starting string. The $PIECE function ($P) in M uses a delimiter to extract a substring from the original string. The syntax can be coded $P(VARIABLE,DELIMITER,FROM PIECE,TO PIECE). If coded with just the VARIABLE and DELIMITER, $P(VARIABLE,DELIMITER), the piece extracted is assumed to be the first piece delimited by the DELIMITER. Another form of this function is $P(VARIABLE,DELIMITER,FROM PIECE) which returns the piece number from the original string delimited by DELIMITER. A certain piece number of a string can be set to a specific value using the SET $P(VARIABLE,DELIMITER,PIECE NUMBER)= syntax. This enables the programmer to change a specific substring of characters in the original string.

In COBOL the PICTURE clause provides the capability for formatting numeric fields with editing characters. COBOL usually needs two specifications for the same field, the unedited field used for storage and the edited field which is

used for displaying fields in reports or on the screen. In M the $FNUMBER function ($FN) formats numbers for positive and negative signs or suppresses the signs, inserts commas, inserts trailing signs, places negative numbers in parentheses and optionally inserts decimal places. Two declarations of the same field, for storage and display, are not necessary in M.

COBOL uses the JUSTIFIED RIGHT clause after the PICTURE clause in the data-item description of a data item to override the normal left justification. The JUSTIFIED RIGHT clause works only for the receiving field in a COBOL MOVE. M uses the $JUSTIFY function ($J) to right justify either a number or character string in a user-specified field width. An option exists for inserting decimal places. The syntax can be either: $J(VARIABLE,WIDTH), or alternatively, $J(VARIABLE,WIDTH,DECIMAL).

COBOL-85 increases the number of table dimensions from three to seven. The PERFORM/VARYING statement has been changed to allow a variable number of AFTER clauses to accommodate the increased table dimensions.

```
PERFORM STEP-THRU-TABLE
   VARYING FIRST-INDEX FROM 1 BY 1
   UNTIL FIRST-INDEX > 5
     AFTER INDEX-TWO FROM 1 BY 1
     UNTIL INDEX-TWO > 5.
```

M's local and global arrays can contain any number of subscript levels. M's arrays use a hierarchical tree structure, not a table matrix structure. The $ORDER function will step through an array structure. The $GET function can be used to access an array and will return a null ("""") if there is only a pointer node (no data) in the array. The size of M's global arrays is not restricted by the language, but may be restricted by the operating system.

```
FOR SET I=$O(^TABLE,(TINDEX)) Q:I=""

SET I=$G(^TABLE(TINDEX))
```

M uses the $DATA function ($D) as a status indicator to show if a variable exists, and if it does exist, whether it contains data. There are four results which can be obtained with the $DATA function, 0, 1, 10 and 11. A 0 indicates that the variable does not exist. A 1 indicates that a variable or part of an array contains data, but if it is part of an array it contains no descendants. A 10 indicates that the variable is part of an array, that the node exists and has descendants, but does not contain data. An 11 indicates that the variable is part of an array, that the node exists, has descendants and contains data. The $DATA can be used with the $ORDER function, which traverses a global array, to ascertain that the global or global node exists, has data and has descendants before attempting to traverse it. No corresponding function exists in COBOL.

The $ORDER function ($O) is used to traverse a subscripted global or array reference. It will step through a global at the level of the subscripted reference and returns every subscript which is defined at the level of the subscripted reference. Data is retrieved from the array for the subscripted reference level. To initialize a subscripted level null ("""") is used. Null is also used to end a subscripted level sequence. The $ORDER is a powerful, flexible tool for accessing M's global arrays (M's database). No corresponding function exists in COBOL.

M has the $LENGTH function ($L) which will indicate the length of a variable string in integers or return the number of fields separated by a specified delimiter. The syntax is $L(VARIABLE) which returns the length of VARIABLE, or $L(VARIABLE,DELIMITER) where the number of fields in VARIABLE separated by DELIMITER is returned.

The $FIND function ($F) with the syntax $F(VARIABLE,CHARACTER(S) TO BE FOUND) or $F(VARIABLE,CHARACTER(S) TO BE FOUND, STARTING POINT OF SEARCH). The $FIND will return the "integer corresponding to the next character position in the target string after the found string."[11]

M's CONTAINS operator "[" compares two string expressions. If the left-hand string contains the right-hand string the result is true (1). M's FOLLOWS operator "]" compares two string expressions to ascertain whether the left-hand string follows the right-hand string in the ASCII collating sequence. The values returned are true (1) or false (0). COBOL's GREATER THAN operator ">" compares to FOLLOWS. COBOL does not have a function that compares with M's CONTAINS operator.

M's $ASCII will return the ASCII decimal value for a character. The $CHAR function is the opposite of the $ASCII function and will return the ASCII character of a number.

COBOL-85 standards increased the maximum length of a nonnumeric literal to 160 characters. M's maximum length of a string literal is 255 characters.

Readability

Readability of computer programs has been considered to contribute to the maintainability of programs. Programming structures that were found to contribute to readability were:

"Extensiveness of comments
Extensiveness of blanks in the left margin
Extensiveness of blank lines
Average length of variable names
Number of arithmetic operators per 100 lines
Average number of goto statements per label"[18]

A study of student programmers found that the amount and length of comments was the only factor that proved to be statistically significant when related to actual programming time.[18]

COBOL's English-like language makes its program logic fairly easy to understand and follow even by non-programmers. COBOL programs can be very lengthy, which works against readability. The use of GOTO's in some of the older unstructured code makes programming logic hard to follow. Emphasis on structured code has improved readability standards.

M's readability is a concern because of its abbreviated commands and functions. In older MUMPS code the entire 255 character maximum line length was used in order to enhance operating efficiency on minicomputers, but using the entire line length tended to decrease readability. Block structuring was added to M in order to improve readability. Documentation and capitalization of M commands and functions has been suggested to aid in readability.[16] M's flexibility tends toward unstructured code. In the M community, structured code has been seen as being less efficient. Recent changes to ANSI standards have made structured M code easier and more desirable to write and maintain. The structuredness leads to enhanced readability.

Conclusion

Each language has related strengths and weaknesses inherent to the purpose for which it was developed. Historically, a great number of COBOL data processing shops have existed. Many COBOL shops and programmers have strongly entrenched negative feelings regarding changing their programming language in general and especially regarding MUMPS in particular.[10] Considering that software represents from 50 to 80 percent of overall corporate data processing budgets, it would make good business sense to choose a computer language that contributes to overall productivity and cost efficiency by using system resources effectively. Please see EXHIBIT A for a comparison of a simple COBOL and M program for reading a file (global) and writing a report.

According to John Spillane, who has seventeen years of experience in programming various languages including Assembler, C, COBOL, BASIC, FORTRAN AND MUMPS, in M "there is no need for the Environment Division of COBOL, or the memory allocation of C. MUMPS does not require dimension statements, sorting routines, record size or field size decisions, opening and closing files, or data type declarations and conversions that most 3GLs require. The programmer need not worry about disk or memory space compaction as the MUMPS integrated database handles that transparently. Database access is handled for the programmer. The programmer need not worry about sequential files versus random files versus ISAM. Database and variable names are self-documenting. Because of these advantages, applications can be developed much more quickly than in other languages."[20]

There have been predictions that the programming field in the United States will eventually go through a restructuring and downsizing in order to meet foreign competition.[22] The issues at stake in the programming field are the same as they were in the manufacturing sector a few years ago: cost, productivity, efficiency and quality. M traditionally runs on smaller, less expensive minicomputers or microcomputers, but even on mainframes it uses fewer system resources. It utilizes less disk space and runs in less time than COBOL applications. Development time is less for M systems than for COBOL systems. All of these factors relate to cost, productivity, and efficiency. Quality is an issue that individuals, both employees and managers, must be concerned about in order to meet the needs of customers and clients. Utilizing a language that enhances productivity and efficiency at a lower cost, would leave more time to devote to quality issues.

Effectively utilizing various computer languages to harness their individual strengths is the trend of future system

environments. M has demonstrated strengths in the cost, productivity and efficiency areas in addition to an ability to share data with other systems. Both M and COBOL have evolved since their initial development. M's strengths were shown in its superior performance on the benchmark tests. Its flexibility and efficiency in regard to operating environments, program structure, sorting, and data types were demonstrated. M expedites application development and enhances programmer productivity.[19] Quality and cost issues indicate that M should be aggressively promoted, especially as an alternative to COBOL.

REFERENCES

1. Alonzo, Casimir M. "System Performance: A Benchmark Study on MUMPS and Other Systems." *MUG Quarterly*. Vol. 13, No. 3. (1983): 13-16.

2. Barnes, Jeff. "A Case for MUMPS." *DBMS*. Vol. 2, No. 6 (June 1989): 58-65.

3. Bozman, Jean S. "Question everything!" *Computerworld*. Vol. 27, No. 1 (28 Dec. 1992 - 4 Jan. 1993): 6-7.

4. Bradley, James. *Introduction to Database Management in Business*. New York: Rinehart and Winston, 1987.

5. Corman, Jared S. and Pasco, Daniel. "SQL and MUMPS: The Right Combination for Business Applications." *MUMPS Computing*. Vol. 22, No. 1 (February 1992): 24-27.

6. "DSM Gets Top Marks in TPC Benchmark." *MUMPS News*. Vol. 7, No. 4 (October 1990).

8. Herring, Barry. "TPC Benchmark A - An Industry Standard Performance Test." *MUG Quarterly*. Vol. 29, No. 1, (June 1990): 84-86.

9. Holst, Sebastian. "Integrating Digital Standard MUMPS (DSM) and the FOCUS 4GL." *MUMPS Computing*. Vol. 22, No. 1. (February 1992): 29-31.

10. Kohun, Frederick G. *Technological Innovation and Diffusion in Medical Software: A Case Study of MUMPS*. Diss. Carnegie-Mellon University, 1990. UMI, 1990. DA9026767.

11. Lewkowicz, John. *The Complete MUMPS*. Englewood Cliffs, New Jersey: Prentice Hall Inc., 1989.

12. Munnecke, Thomas. "A Linguistic Comparison of MUMPS and COBOL." National Computer Conference, AFIPS, (1980): 723-729.

13. Munnecke, T., Walters, R. F., Bowie, J., Lazarus, C. B., and Bridger, D. A. "MUMPS: Characteristics and comparisons with other programming systems." *Medical Informatics*. Vol. 2, No. 3, (1977): 173-176.

14. Murach, Mike and Noll, Paul. *Structured ANS COBOL*. Fresno, California: Mike Murach & Associates, Inc., 1987.

15. Philippakis, A. S. and Kazmier, Leonard J. *The New COBOL An Illustrated Guide*. New York: McGraw-Hill Book Company, 1986.

16. Ravenhill, Kevin and Schindler, Karen. "Efficiency versus Structure - is it a Compromise?" *MUMPS Computing*. Vol. 22, No. 3, (June 1992): 74-80.

17. Rex, John. "A Touch of MUMPS." *Computer Language*. Vol. 6, No. 11, (November 1989): 55-67.

18. Richmond, Joseph R. *Software Maintainability Metrics for MUMPS Programs*. Diss. The University

19. Smith, Jill Y. and Harvey, William J. "MUMPS is Germinating Business Productivity." *Journal of Systems Management*. Vol. 39, No. 4, (April 1988): 26-31.

20. Spillane, John. "MUMPS in Credit Unions." *MUMPS Computing*. Vol. 22, No. 1, (February 1992): 18-21.

21. Welburn, Tyler. *Structured COBOL Fundamentals and Style*. Palo Alto, California: Mayfield Publishing Company, 1986.

22. Yourdon, Edward. *Decline & Fall of the American Programmer*. Englewood Cliffs, New Jersey: ard. Yourdon Press, 1992.

EXHIBIT A

M PROGRAM

```
11:30 AM  14-JAN-93
SALES COMMISSION
SALESCOM ; NEW PROGRAM [ 01/14/93   11:29 AM]
 ; SALES COMMISSION REPORT
 ; ASSUME A GLOBAL OF
 ; ^SCOM(SNUM)=RCD_^_REGIN_^_TERR_^_SNAME_^_CPCT_^_UNITS_^_CAMT
 D INIT
 D MAIN
 Q
MAIN ; main routine
 S SNUM=""
 F  S SNUM=$O(^SCOM(SNUM)) Q:SNUM=""  D
 . S DATA=^SCOM(SNUM),RCD=$P(DATA,DL,1),REGIN=$P(DATA,DL,2),TERR=$P(DATA,DL,3),
 SNAME=$P(DATA,DL,4),CPCT=$P(DATA,DL,5),UNITS=$P(DATA,DL,6),CAMT=$P(DATA,DL,7)
 . S FCPCT=$FN($J(CPCT,5,2),",",),FCAMT=$FN($J(CAMT,8,2),",",)
 . W !!?5,REGIN,"-",TERR,?15,SNUM,?20,SNAME,?50,UNITS,?55,FCPCT,"%",?65,"$",FCAMT
 Q
INIT ; initialization routine
 S DL="^"
 Q
```

* THE LABEL HEADINGS IN THE M PROGRAM AND THE DIVISION HEADINGS IN THE COBOL PROGRAM ARE BOLDED FOR DISPLAY PURPOSES ONLY. THIS WOULD NOT OCCUR IN A PROGRAM LISTING.

EXHIBIT A

COBOL PROGRAM

```
IDENTIFICATION DIVISION.
PROGRAM-ID.      SALES-COMM-REP.
*THIS PROGRAM WILL PRINT A SALES COMMISSION REPORT FROM A SALESPERSON FILE
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
      SELECT SALESPERSON-FILE
           ASSIGN TO INFILE.
      SELECT SALES-COMM-FILE
           ASSIGN TO OUTFILE.
DATA DIVISION.
FILE SECTION.
FD  SALESPERSON-FILE
    RECORD CONTAINS 80 CHARACTERS
    DATA RECORD IS INPUT-RECORD.
01  INPUT-RECORD.
      05  REC-CODE              PIC 9(2).
      05  REG-IN                PIC X(2).
      05  TERRITORY-IN          PIC 9(4).
      05  SALESPERS-NUMB-IN     PIC 9(3).
      05  SALESPERS-NAME-IN     PIC X(26).
      05                        PIC X(1).
      05  COMM-PRCNT-IN         PIC 9(2)V99.
      05                        PIC X(5).
      05  UNITS-SOLD-IN         PIC S9(3).
      05                        PIC X(3).
      05  COMM-AMT-IN           PIC 9(5)V99.
      05                        PIC X(20).
FD    SALES-COMM-FILE
      RECORD CONTAINS 131 CHARACTERS
      DATA RECORD IS OUTPUT-RECORD.
01  OUTPUT-RECORD.
      05                        PIC X(5).
      05  REG-OUT               PIC X(2).
      05  HYPHEN-OUT            PIC X(1).
      05  TERRITORY-OUT         PIC 9(4).
      05                        PIC X(3).
      05  SALESPERS-NUMB-OUT    PIC 9(3).
      05                        PIC X(3).
      05  UNITS-SOLD-OUT        PIC ----.
      05                        PIC X(3).
      05  COMM-PRCNT-OUT        PIC ZZ.99.
      05  PRCNT-SIGN-OUT              PIC X(1).
      05                        PIC X(3).
      05  COMM-AMT-OUT          PIC $ZZ,ZZZ.99.
      05                        PIC X(55).

WORKING-STORAGE SECTION.
01  WS-EOF                     PIC X(3)  VALUE "NO".
```

Continued on next page

```
PROCEDURE DIVISION.
MAIN-ROUTINE.
       OPEN INPUT SALESPERSON-FILE
            OUTPUT SALES-COMM-FILE.
       READ SALESPERSON-FILE
           AT END MOVE "YES" TO WS-EOF.
       PERFORM MOVE-WRITE-PROCESS
           UNTIL WS-EOF = "YES".
       CLOSE SALESPERSON-FILE
             SALES-COMM-FILE.
       STOP RUN.
MOVE-WRITE-PROCESS.
       MOVE SPACES TO OUTPUT-RECORD.
       MOVE REG-IN TO REG-OUT.
       MOVE "-" TO HYPHEN-OUT.
       MOVE TERRITORY-IN TO TERRITORY-OUT.
       MOVE SALESPERS-NUMB-IN TO SALESPERS-NUMB-OUT.
       MOVE SALESPERS-NAME-IN TO SALESPERS-NAME-OUT.
       MOVE UNITS-SOLD-IN TO UNITS-SOLD-OUT.
       MOVE COMM-PRCNT-IN TO COMM-PRCNT-OUT.
       MOVE "%" TO PRCNT-SIGN-OUT.
       MOVE COMM-AMT-IN TO COMM-AMT-OUT.
       WRITE OUTPUT-RECORD
           AFTER ADVANCING 2 LINES.
       READ SALESPERSON-FILE
           AT END MOVE "YES" TO WS-EOF.
```