https://ntrs.nasa.gov/search.jsp?R=19730010491 2020-03-17T08:25:28+00:00Z

N73-19248

COST ADVANTAGES OF AN INTEGRATED DOCUMENTATION APPROACH

William O. Felsman
Litton Systems, Inc.

Interactive use of on-line computing terminals is a method of increasing importance in developing or updating programs. Two major advantages are realizable: the turnaround time is decreased by at least one order of magnitude, and the programmer learns to treat the computer not only as a computational device but also as a combination blackboard and library. All additions and updatings are consequently performed directly on disc storage, and the completed program is thus available only in the raw form in which it was developed.

Program cleanup is required before formal documentation. TIDY and SWAP are two programs that assist this function. Figure 1 shows a subroutine that is representative of the kind of source code that can result when using on-line, interactive programming techniques, particularly if full advantage is made of the on-line system by assigning several programmers to related portions of the same task.

The TIDY program reassigns statement numbers in ascending order, with the base number and number increment being defined by the operator. It also generates a standardized, closed-up source language format. Figure 2 shows the RAW subprogram after being processed by TIDY.

```
      SUBROUTINE    SUB3( I)
       DIMENSION  IARRY(32 )
      IEMP1=I
      IEMP2=   1073741824
         IF(I)77, 10, 10
77    I= I +2147483647+  1
       IARRY( 1)= 1
       GO TO  11
 10   IARRY( 1)=0
  11 DO   16    J=1,31
      IF (I  - IEMP2)12,88,88
  12 IARRY(J+ 1)=0
      GO TO 90
88    IARRY(J+ 1)= 1
       I=I-IEMP2
 90   IEMP2=IEMP2/2
  16 CONTINUE
      I= IEMP1
      WRITE(6,800) (IARRY(J1) ,J1=1,32 )
 800 FORMAT(1X, 4(4I1, 1X, 4I1 ,2X ))
      RETURN
      END
```

Figure 1.–RAW subprogram.

Two further transformations are useful for improving the program legibility. First, the equal, plus, and minus operators can be set off by blanks and the variable names can be replaced by others of increased semantic content. Figure 3 shows the TIDY'ed program after being thus processed by SWAP. The format is more pleasing, and the variables now indicate the function they serve. For example, the name of the subroutine BISHOW shows some relation to its function of generating a bit-by-bit printout of the contents of the named variable, and so on.

PREDOCUMENTED SUBROUTINES

The use of predocumented subroutines in the development of a new

PRECEDING PAGE BLANK NOT FILMED

```
      SUBROUTINE SUB3(I)
      DIMENSION IARRY(32)
      IEMP1=I
      IEMP2=1073741824
      IF(I)10,12,12
   10 I=I+2147483647+1
      IARRY(1)=1
      GO TO 14
   12 IARRY(1)=0
   14 DO 22 J=1,31
      IF(I-IEMP2)16,18,18
   16 IARRY(J+1)=0
      GO TO 20
   18 IARRY(J+1)=1
      I=I-IEMP2
   20 IEMP2=IEMP2/2
   22 CONTINUE
      I=IEMP1
      WRITE(6,24)(IARRY(J1),J1=1,32)
   24 FORMAT(1X, 4(4I1, 1X, 4I1 ,2X ))
      RETURN
      END
```

Figure 2.–TIDY'ed program.

```
      SUBROUTINE BISHOW(NUMBER)
      DIMENSION MEMBIT(32)
      NUMSAV = NUMBER
      NEXT = 1073741824
      IF(NUMBER)10,12,12
   10 NUMBER = NUMBER + 2147483647 + 1
      MEMBIT(1) = 1
      GO TO 14
   12 MEMBIT(1) = 0
   14 DO 22 ICOUNT = 1,31
      IF(NUMBER - NEXT)16,18,18
   16 MEMBIT(ICOUNT + 1) = 0
      GO TO 20
   18 MEMBIT(ICOUNT + 1) = 1
      NUMBER = NUMBER - NEXT
   20 NEXT = NEXT/2
   22 CONTINUE
      NUMBER = NUMSAV
      WRITE(6,24)(MEMBIT(J1),J1 = 1,32)
   24 FORMAT(1X,4(4I1,1X,4I1,2X))
      RETURN
      END
```

Figure 3.–TIDY'ed and SWAP'ed program.

Table 1.–List of Common Subroutines

Compiler	Assembler	LOG2	SADL	SWAP	TIDY	Subroutine	Definition
X	X			X	X	PAGE	Page control
		X	X	X	X	INBUF	Source language read control
				X	X	OUTBUF	Source language standard output format
X	X	X	X	X	X	MOVER	Right adjust name
X	X	X				MOVEL	Left adjust name
	X			X		MVLEFT	Left adjust name, return spaces shifted
	X					MVRITE	Right adjust name, return spaces shifted
	X	X		X	X	ILSHFT	Shift name specified number of places to the left
	X	X	X	X	X	IRSHFT	Shift name specified number of places to the right
	X			X		IFXPT	Alpha to integer conversion
	X	X	X	X	X	DLIMIT	Delimit a source statement
	X	X	X	X	X	PACK3P	Terse number equivalence to variable name
	X	X	X	X	X	PACKP3	Reassemble names after delimiting
	X			X	X	UNPACQ	A4 format to A1 format
	X					ORDER	Indirect reference ordering of data

program is a well-known way to greatly reduce the attendant documentation effort at the same time that it reduces the program development time. Table 1 shows the extent to which predocumented subroutines were useful in the development of the operational programs discussed in this paper. The effect is similar to the use of a special higher order language, except that the more powerful operations are defined by subprograms rather than operators.

METAPROGRAM CONCEPT

Considerable additional advantage can be obtained if programs are written in a generic manner. For example, in an assembler program for an avionics computer, completion of the truncated operand addresses is normally a function of the attendant instruction. The alternatives might be to use the most significant bits of the instruction counter, to use a maximal length base address register, or to use a minimal base address register. If the usage for each instruction is written into the source code, then modification of the program to perform the assembly function for a second computer requires not only that the source code be redeveloped but also that the documentation be rewritten both at the program level and the user's manual level.

If, on the other hand, the basic program is written to provide for all of these potential choices and the branch chosen for a particular command is determined by a data set including the command mnemonic and a code defining the method of address completion, then this portion of the program needs no rewriting or redocumentation when the target computer is modified or a new target computer developed. Change of the data-set entry table is all that is required, together with the attendant minimal documentation.

This method of identifying significant parameters in a class of programs and then writing a generic program to accommodate these parameters in a defining data set is known as a "metaprogram" approach. That is, the data set is itself in effect a program, written in some very simple interpretive language and consequently has become known as the metaprogram.

Figures 4 and 5 show the metaprograms used for TIDY and SWAP.

The TIDY metaprogram is categorized by an identifying operator and several parameters. These define the start and finish locations within a FORTRAN statement of the series of places where statement numbers occur in that statement. The main program is thus

```
   DO   2   2
   GO   3   3   1
   GO   4  -4
 GOTO   2   2   1
 GOTO   3  -4
   IF  -1  -1   1
   IF  -5  -1
CYCLE   2   2
 READ   5   5
WRITE   5   5
```

Figure 4.–TIDY metaprogram.

```
GO=GO DELIMIT
TO=TO DELIMIT
CALL=CALL DELIMIT
REAL=REAL DELIMIT
INTEGER=INTEGER DELIMIT
SUBROUTINE=SUBROUTINE DELIMIT
DIMENSION=DIMENSION DELIMIT
DOUBLE=DOUBLE DELIMIT
PRECISION=PRECISION DELIMIT
DO=DO DELIMIT2
COMPLEX=COMPLEX DELIMIT
COMMON=COMMON DELIMIT
CYCLE=CYCLE DELIMIT
GOTO=DELIMITL GO DELIMIT TO DELIMIT DELIMITR
EQUIVALENCE=EQUIVALENCE DELIMIT
FUNCTION=FUNCTION DELIMIT
```

Figure 5.–SWAP metaprogram.

specifically directed to the locations where replacement is to occur. Additional statement types can be processed by adding to the metaprogram with no change in either the main program or the documentation associated with it.

The SWAP metaprogram is equally simple. The mnemonic to be replaced occurs to the left of the equal sign, and the replacing mnemonic sequence is to the right of the equal sign. Additional control parameters are included at the right. Thus, DELIMIT indicates a blank is to be inserted after each occurrence of a new variable in the output, and DELIMIT2 indicates that blanks are to be placed after both the replacing variable and the next succeeding variable. Additional control commands close up blocks, remove parentheses, and perform other functions.

This metaprogram is also open-ended, and additional statements in any of several higher order languages can be accommodated by appropriate descriptions in the metaprogram.

In fact, the data representing the desired mnemonic exchange is also treated as a simple continuation of the normal metaprogram.

```
STA1  CLA2  18001212  18001010
STA1        18001010
LDQ1  STA2  24001212  24001010
LDQ1        24001010
CLA2  ADD1  25012121  28001010
CLA2  SUB1  29012121  28001010
CLA1  STA2  28001212  28001010
CLA1        28001010
CLA2  MPY1  19012121  28001010
INC1  TRZ2  14001515  14001010
STQ1  STA2  16001212  16001010
STQ1        16001010
ADD1  STA2  26001212  26001010
ADD1        26001010
SUB1  STA2  27001212  27001010
SUB1        27001010
CLA2  ANA1  31012121  28001010
ANA1        31001010  31001010
CLA2  ORA1  30012121  28001010
ORA1        30001010  30001010
CLA2  DIV1  17012121
DIV1        17001010  17001010
CLA2  TRA1  00012323  00003010
TRA1        00003010
TNZ1  CLA2  01003232  01003010
TNZ1        01003010
TZA1  CLA2  05003232  05003010
TZA1        05003010
TPA1  CLA2  06003232  06003010
TPA1        06003010
```

Figure 6.–Typical assembler metaprogram (partial).

Figure 6 shows a portion of a metaprogram for an assembler. It consists of a defining mnemonic or mnemonic pair, plus a series of numerals that define the transfers within the main program which control the development of the assembled code for that instruction mnemonic. The assembler program is very nearly invariant for a wide class of computers, consequently documentation of the main program can here, too, be unchanged with new applications. However, the interpretation of the metaprogram by the programmer requires a computer assist to documentation.

Figure 7 shows the metaprogram as processed for semantic clarity, and figure 8 shows a portion of the documentation that relates the metaprogram controls to the operation of the computer arithmetic unit. The matrix documentation shown in figure 8 has been developed to reduce the difficulty in the handling, updating, and correction of bulk data. The matrix documentation program accepts data as a sequential input stream and formats it in both vertical and horizontal directions, with text hyphenation where applicable. Program control cards direct the number and width of the columns. Thus, each entry in any column is separately modifiable, and the program adjusts the full updated data input to maintain the format.

INTEGRATED DOCUMENTATION SAVINGS

Two examples of the cost of documentation are shown in table 2.

In each case, the compiler and assembler documentation for a given computer required full page counts of 140 and 95, respectively. However, for all successive applications to

		NON ZERO SECONDARY			DIRECT		INDIRECT			ZERO SECONDARY			DIRECT		INDIRECT		
PRI	SEC	NUMB	TYPE	M	PRI	SEC	PRI	SEC	SPR	NUMB	TYPE	M	PRI	SEC	PRI	SEC	SPR
STA1	CLA2	18	0	0	1	2	1	2	0	18	0	0	1	0	1	0	0
STA1		18	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
LDQ1	STA2	24	0	0	1	2	1	2	0	24	0	0	1	0	1	0	0
LDQ1		24	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
CLA2	ADD1	25	0	1	2	1	2	1	0	28	0	0	1	0	1	0	0
CLA2	SUB1	29	0	1	2	1	2	1	0	28	0	0	1	0	1	0	0
CLA1	STA2	28	0	0	1	2	1	2	0	28	0	0	1	0	1	0	0
CLA1		28	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
CLA2	MPY1	19	0	1	2	1	2	1	0	28	0	0	1	0	1	0	0
INC1	TRZ2	14	0	0	1	5	1	5	0	14	0	0	1	0	1	0	0
STQ1	STA2	16	0	0	1	2	1	2	0	16	0	0	1	0	1	0	0
STQ1		16	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
ADD1	STA2	26	0	0	1	2	1	2	0	26	0	0	1	0	1	0	0
ADD1		26	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
SUB1	STA2	27	0	0	1	2	1	2	0	27	0	0	1	0	1	0	0
SUB1		27	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
CLA2	ANA1	31	0	1	2	1	2	1	0	28	0	0	1	0	1	0	0
ANA1		31	0	0	1	0	1	0	0	31	0	0	1	0	1	0	0
CLA2	ORA1	30	0	1	2	1	2	1	0	28	0	0	1	0	1	0	0
ORA1		30	0	0	1	0	1	0	0	30	0	0	1	0	1	0	0
CLA2	DIV1	17	0	1	2	1	2	1	0	0	0	0	0	0	0	0	0
DIV1		17	0	0	1	0	1	0	0	17	0	0	1	0	1	0	0
CLA2	TRA1	0	0	1	2	3	2	3	0	0	0	0	3	0	1	0	0
TRA1		0	0	0	3	0	1	0	0	0	0	0	0	0	0	0	0
TNZ1	CLA2	1	0	0	3	2	3	2	0	1	0	0	3	0	1	0	0
TNZ1		1	0	0	3	0	1	0	0	0	0	0	0	0	0	0	0
TZA1	CLA2	5	0	0	3	2	3	2	0	5	0	0	3	0	1	0	0
TZA1		5	0	0	3	0	1	0	0	0	0	0	0	0	0	0	0
TPA1	CLA2	6	0	0	3	2	3	2	0	6	0	0	3	0	1	0	0
TPA1		6	0	0	3	0	1	0	0	0	0	0	0	0	0	0	0

Figure 7.—Processed assembler metaprogram (partial).

CODE	MNEMONIC	INSTRUCTION CONDITIONS M1 AND M2 FIELD USED NO INDIRECT ADDRESS	INSTRUCTION CONDITIONS M1 FIELD ONLY NO INDIRECT ADDRESS	INSTRUCTION CONDITIONS M1 AND M2 FIELD USED INDIRECT ADDRESSING	INSTRUCTION CONDITIONS M1 FIELD ONLY INDIRECT ADDRESSING
0001	TRA M1 CLA M2	(M2+BARL) --> A PACK(PC,BAR(7))-->DED M1+PCS --> PC Q --> Q BARL --> BARL BARS --> BARS	PACK(PC,BAR(7))-->DED M1+PCS --> PC Q --> Q BARL --> BARL BARS --> BARS	((M2+BARL)) --> A PACK(PC,BAR(7))-->DED M1+PCS --> PC Q --> Q BARL --> BARL BARS --> BARS	PACK(PC,BAR(7))-->DED (M1+BARS) --> PC Q --> Q. BARL --> BARL BARS --> BARS
0002	TNZ M1 CLA M2	M1+PCS -->PC (A NON 0) (M2+BARL) --> A PACK(PC,BAR(7))-->DED Q --> Q BARL --> BARL BARS --> BARS	M1+PCS -->PC (A NON 0) PACK(PC,BAR(7))-->DED Q --> Q BARL --> BARL BARS --> BARS	M1+PCS-->PC (A NON 0) ((M2+BARL)) --> A PACK(PC,BAR(7))-->DED Q --> Q BARL --> BARL BARS --> BARS	(M1+BARS)-->PC(A NO 0) PACK(PC,BAR(7))-->DED Q --> Q BARL --> BARL BARS --> BARS
0003	LBR M2 TRA M1	M2 --> BAR (7 BITS) M1+PCS(NEW) --> PC Q --> Q BARL(NEW) --> BARL BARS(NEW) --> BARS PC+1 --> PC	(M1+PCS(OLD)) --> PC Q --> Q BARL(OLD) --> BARL BARS(OLD) --> BARS PC+1 --> PC	(M2+BARL) --> BAR(7) M1+PCS(NEW) --> PC Q --> Q BARL(NEW) --> BARL BARS(NEW) --> BARS	((M1+BARS(OLD)))-->PC Q --> Q BARL(OLD) --> BARL BARS(OLD) --> BARS
0004	SPC M1 CLA M2	(DED)--> (M1+BARS) (M2+BARL) --> A Q --> Q BARL --> BARL BARS --> BARS PC+1 --> PC	(DED)--> (M1+BARS) Q --> Q BARL --> BARL BARS --> BARS PC+1 --> PC	(DED)--> (M1+BARS) ((M2+BARL)) --> A Q --> Q BARL --> BARL BARS --> BARS PC+1 --> PC	(DED)--> ((M1+BARS)) Q --> Q BARL --> BARL BARS --> BARS PC+1 --> PC
0005	STA M2 TRA M1	(A) --> (M2+BARL) PACK(PC,BAR(7))-->DED M1+PCS --> PC Q --> Q BARL --> BARL BARS --> BARS	PACK(PC,BAR(7))-->DED M1+PCS --> PC Q --> Q BARL --> BARL BARS --> BARS	(A) --> ((M2+BARL)) PACK(PC,BAR(7))-->DED M1+PCS --> PC Q --> Q BARL --> BARL BARS --> BARS	PACK(PC,BAR(7))-->DED (M1+BARS) --> PC Q --> Q BARL --> BARL BARS --> BARS
0006	TZA M1 CLA M2	(M2+BARL) --> A PACK(PC,BAR(7))-->DED M1+PCS-->PC (A=0) Q --> Q BARL --> BARL BARS --> BARS	PACK(PC,BAR(7))-->DED M1+PCS-->PC (A=0) Q --> Q BARL --> BARL BARS --> BARS	((M2+BARL)) --> A PACK(PC,BAR(7))-->DED M1+PCS -->PC (A=0) Q --> Q BARL --> BARL BARS --> BARS	PACK(PC,BAR(7))-->DED (M1+BARS)-->PC (A=0) Q --> Q BARL --> BARL BARS --> BARS
0007	TPA M1 CLA M2	(M2+BARL) --> A PACK(PC,BAR(7))-->DED M1+PCS -->PC (A=+) Q --> Q BARL --> BARL BARS --> BARS	PACK(PC,BAR(7))-->DED M1+PCS -->PC (A=+) Q --> Q BARL --> BARL BARS --> BARS	((M2+BARL)) --> A PACK(PC,BAR(7))-->DED M1+PCS -->PC (A=+) Q --> Q BARL --> BARL BARS --> BARS	PACK(PC,BAR(7))-->DED (M1+BARS)-->PC (A=+) Q --> Q BARL --> BARL BARS --> BARS

Figure 8.—Matrix documentation of assembler metaprogram (partial).

Table 2.–Documentation Page Count

Documentation stage	Compiler	Assembler
Theory of operation	70	40
Flowcharts	30	30
Description of charts	30	20
Metaprogram description	10	5
Total	140	95

Table 3.–Program Organization Statistics, Source Code

Name	Basic program length (lines)	Total program length (lines)	Development time (man-months)	Basic program[a]	Separable program-peculiar subroutines[a]	Data set[a]	Predocumented subroutines[a]
TIDY (statement number reorder)	229	686	0.75	0.334	0.052	0.015	0.599
SADL (reliability model)	475	816	1.5	.590	.025	.007	.378
LOG2 (logic simulator)	515	857	1.5	.602	.000	.003	.395
SWAP (mnemonic exchange)	210	637	.5	.330	.000	.024	.646
Memory allocator	587	887	3	.662	.025	.016	.298
Assembler	829	2193	5.5	.378	.077	.063	.282
Compiler	1216	1690	24	.720	.063	.200	.017

[a]Proportions of total programs.

differing computers, the compiler required only a new description of the metaprogram, and the assembler, which was less completely organized into a generic program, required a new metaprogram description plus about 30 percent of the other page counts.

Some indication of the savings to be gained by the use of predocumented subroutine and metaprogram data-set techniques can be seen in table 3, which gives the relative usage of these methods for a variety of programs. Documentation savings run typically from 30 to 60 percent on the initial program development. However, the use of subprograms has a major effect upon the program cost itself. Table 3 tabulates the number of lines of source code for the programs investigated together with the development time for these programs. These programs were developed by a group of five people, of consistent skill level, working both individually and as members of small teams. A plot of these data is shown in figure 9.

The most significant aspect of the graph is that it shows an exponential growth of development time with the length of the basic program. The basic program consists of the main program plus such program-peculiar subroutines as are conceptually involved with the main program in a highly complex manner. Thus, implementation of an integrated

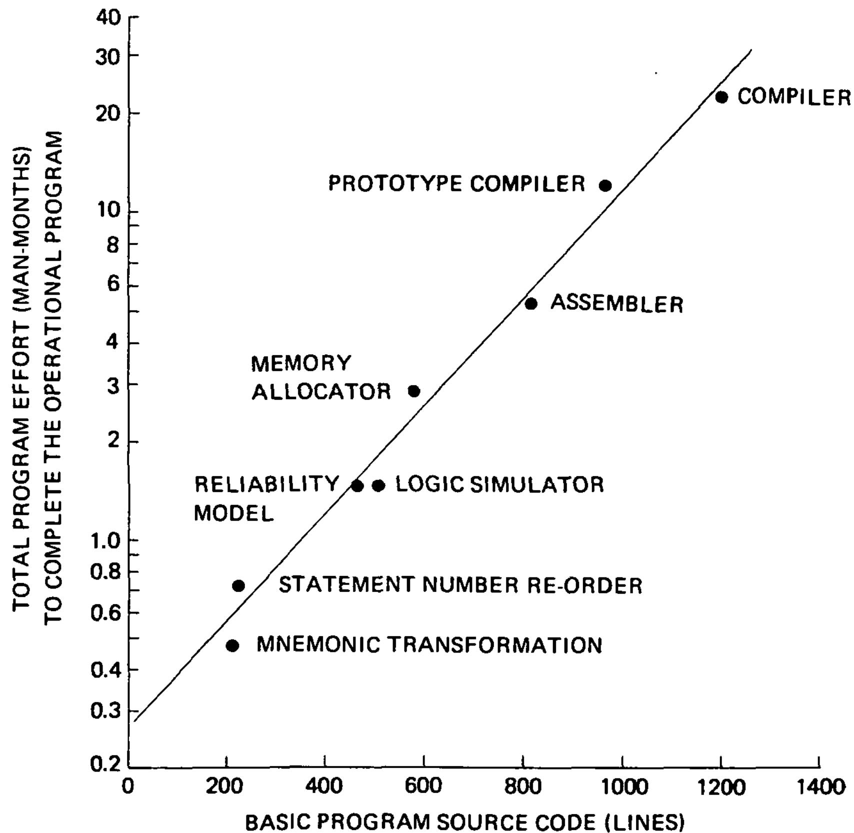

Figure 9.–Program development time.

documentation approach, which suggests the use of small, preferably predocumented, sub-program elements is not inconsistent with the program design cycle, which also shows greater efficiency in the time of program development when appropriate organization permits the size of the basic program to be reduced.

MODIFICATION

Figure 10 shows the advantages resulting from the use of the integrated documentation approach when it was necessary to modify an existing assembler to accommodate a new computer. Two basic programs were available for modification, one with both subroutines and a metaprogram, the other being predominantly large program elements. Eighty-five percent of the source code from the integrated approach was applicable, whereas only 40 percent of the unitized code could be reused.

Figure 10 also shows the costs associated with the two modifications. The unitized code changeover was estimated at 15 man-months. The actual cost to update the program using an integrated documentation approach was 2 man-months.

Further cost savings are shown in table 4. Here, the effect of the metaprogram is detailed. The cost of modification from one target computer to another is compared to the

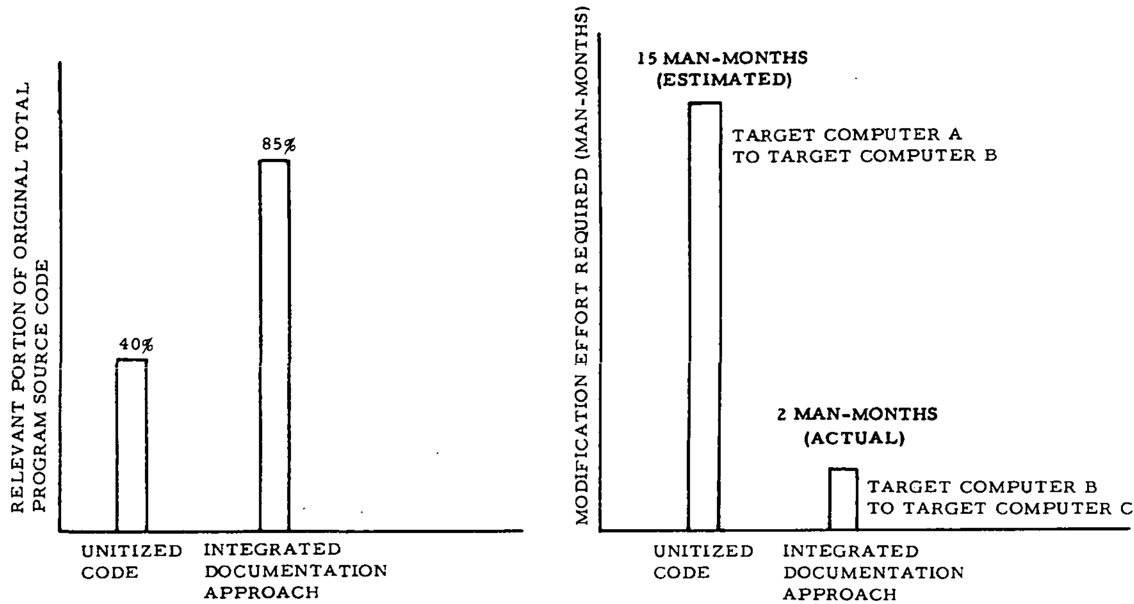

Figure 10.—Program modification comparison (assembler program).

Table 4.—Integrated Documentation Effects on Program Modification

Program type	Data set proportion in source code	Rate of cost of modification to new program design
Compiler	0.20	0.1
Assembler	.06	.3

cost of a new program design for the new target computer. Examples are given for a compiler and an assembler.

The compiler, which makes extensive use of a metaprogram, can be modified to accommodate a new target computer at one-tenth the cost of writing a new compiler program. Modifications are almost entirely to the metaprogram. The assembler, which uses a less well-developed metaprogram but which does make extensive use of common subroutines, can be modified to accommodate a new target computer at about one-third the cost of developing a new assembler program.

RUNNING TIME

Run time of production programs developed using the integrated documentation approach would appear at first consideration to be somewhat higher than the run times of similar programs written in unitized code. This is because the use of prepackaged subroutines implies a close, but not exact, fit between the requirement and the package and

because of the extra time required to execute the branch statements implied by metaprogram techniques.

However, the integrated documentation approach tends to segment a program into functionally consistent elements with minimal communication required between them and consequently parallels good programming practice. The result is that the total program size is somewhat reduced, values from 10 to 30 percent being typical. Smaller programs, using less core, generally are charged a lesser main frame usage rate. This lesser rate compensates for the longer execution time.

CONCLUSION

An integrated documentation approach using predocumented subroutines and metaprogram techniques is a very efficient means of not only generating the relevant documentation but also of reducing program development costs.

DISCUSSION

MEMBER OF THE AUDIENCE: Do you use this approach on all your programs? In other words, do you think that all programs are divisible into small metaprograms?

FELSMAN: Yes, they are divisible, but we do not always do it because occasionally you run into someone who wants something in a hurry. Then we simply write in a standard fashion as rapidly as we can. It is a one-of-a-kind thing, and we do not worry about documentation. But for big problems like compilers, assemblers, and memory allocators, we go through the process and do indeed break it out, use our regular subroutines, and always write data-set-wise or metaprogram-wise. It is much more convenient.

MEMBER OF THE AUDIENCE: I think your presentation answered the question raised by Gridley yesterday about whether we should put emphasis on subroutines or smaller parts. At that time we did not really respond to his question on the panel, and I think we should have because it is an asset not only in developing programs but in distributing programs to other people to use. If you are going to use an entire program without modifying it, fine. But when you develop programs, I think your point is valid that it takes less time to develop a new program from well-documented subroutines or metaprograms than it does to modify an existing program to make it work on a computer other than the one for which the program was written.

MEMBER OF THE AUDIENCE: You implied that this was for a given set of target computers.

FELSMAN: Yes. In this case they happen to be all airborne computers.

MEMBER OF THE AUDIENCE: Have you considered it for a general-purpose type of computer in a general-purpose environment?

FELSMAN: As far as I can tell, we have investigated other military computers like the AN/UYK-7, which is a very powerful floating-point machine very similar to some of our commercial machines, and we wrote a metaprogram for that using this compiler. I think the answer to your question is yes, although not unequivocally.

MEMBER OF THE AUDIENCE: Presume a manufacturer has provided this to us, but our problem is related to application programs. Can this be applied there?

FELSMAN: You have a more difficult task because you have to find the common parameters from application to application. If you can find any, you can do it. If they are not immediately evident, maybe you cannot do it.