SOFTWARE DOCUMENTATION SUPPORT

Kerry M. Atchinson
Boeing Military Airplane Company
Wichita, Kansas

ABSTRACT

Generation and support of documentation accompanying software development is historically a low-efficiency, high-cost undertaking. Frequently the situation arises where a choice must be made to fulfill documentation requirements and incur cost and schedule overruns, or to complete the software product in a reasonable time and deliver less than adequate documentation. The primary difficulties are efficient generation, quick and thorough update, and document correlation on large projects. Automated methods can alleviate these problems by (1) application of word processing systems to the generation and editing of descriptive text, (2) use of a data base manager-type control of system interface definition and document correlation, (3) use of pseudo-code for first-time design and flowchart generation, and (4) the use of special purpose software tools to perform analysis of code for flowchart and module interface updating.

Documentation of software can be performed on several levels of complexity. The lowest level is perhaps the basic user's guide that accompanies small software items or those packages where a detailed knowledge of the software internals is unneeded or undesirable. This guide is usually textual, giving only installation and usage instructions. A higher level of complexity is encountered when system descriptions are included, as may be the case with a vendor's operating system. The user in these cases is given needed information on the interplay of the system's elements and possibly a *fair amount of detail on the internal workings of the system in addition* to the installation and usage information. Still omitted are the detailed design information, engineering trade-offs, and associated background information on philosophy, methodology, etc., that must accompany the most complex level of documentation — that given to the military or the government in general. Examples are compliance with the "Part II Specification" called out by USAF Data Item Descriptions DI-E-3120B/M1 Computer Program Product Specifications, and the DI-H-3277/M7 Training Equipment Computer Program Documentation.

The Part II Specification includes computer program descriptions, table and data base descriptions, a real-time cross reference, programmer's notebooks, time and memory allocation tracking data, a Computer Program System (CPS) guide, and of course, any associated vendor manuals. The general contents of the Part II Specification are depicted in Figure 1. Of the documentation items listed, only two items are easily supplied: the listings and vendor manuals.

```
                PART II SPEC
                      :
                      :
+---------------------+
:
:--COMPUTER PROGRAM DESCRIPTIONS
:           :   MODELING
:           :   NARRATIVE
:           :   CHARTS
:           :   LISTINGS
:           :   AS-BUILT TIME AND SIZE
:
:--TABLES AND DATA BASES
:           :   GENERATION
:           :   STRUCTURE
:           :   USAGE
:           :   TIME AND MEMORY CONSTRAINTS
:
:--REAL TIME CROSS REFERENCE
:
:--PROGRAMMER'S NOTEBOOK
:           :   PECULIAR NOTATIONS
:           :   INDIRECT TECHNIQUES
:           :   CODING SIMPLIFICATIONS
:           :   PROBLEMS AND SOLUTIONS
:           :   LANGUAGE CHANGES
:
:--MONTHLY TIME AND SIZE ALLOCATIONS
:
:--CPS GUIDE
:           :   DEVELOPMENT & UPDATE GUIDE
:           :   USER'S GUIDE
:
:--VENDOR MANUALS
```

Figure 1. Part II Specification Structure

On a large software project, the effort required to produce this documentation can be staggering. The Boeing Military Airplane Company (BMAC), in building the B-52/KC-135 Weapons Systems Trainer (WST) prototypes for the USAF, produced approximately 1,000,000 lines of source code. This resulted in roughly 91,000 sheets of non-listing detailed design documentation. This does not include the amount of documentation produced in the generation of functional specifications, implementation specifications, or the programmers' notebooks, CPS guide, and other items of the Part II Spec. Additionally, there were in excess of 7,000 revisions to the source parts comprising the CPS. Assuming that each revision and associated document release required the modification of a minimum *of three sheets (one flowchart, one narrative, and one revision status* sheet), there were at least 21,000 regenerated documentation sheets for a total of 112,000 effective non-listing pages of design documents.

Naturally, there are impediments to the completion of a task of this magnitude. One of the problems with which many organizations must deal is the lack of sufficient numbers of skilled engineers and programmers to perform the designing and coding tasks. Engineering support personnel who should undertake much of the documentation task are therefore frequently utilized to perform in an engineering role. This places a greater burden of document generation upon the engineers and programmers which adds to the schedule impact.

The major issue in documenting software, even with an abundance of designers and support personnel, is the cost. A reasonable cost estimate for a delivered page of documentation is three manhours (MH).[1] It can be assumed that about one MH for one round of editing and revising with an additional 0.5 MH for typing[2] are embedded in this figure. Each additional revision of a page can then be estimated at 1.5 MH. The documentation cost of a WST-sized project could then be expected to be 304,500 MH, or approximately 1,750 man months (MM). This can be further emphasized by noting that a comparison of document generation to code generation shows that documentation accounts directly and indirectly for about 60 percent of project costs as opposed to 40 percent of project costs for coding.[3] Obviously, documentation is a candidate for cost reduction.

How can this cost be reduced? Perhaps the most apparent remedy is the application of word processing to those areas of the documents *that are texual. In-house BMAC experience with word processing* indicates that savings of 60-70 percent are possible when revision and retyping are undertaken through word processing rather than conventional secretarial methods. The 7,000-plus released revisions on WST could be translated to an expected expenditure of 31,500 MH. A 60 percent reduction via word processing brings about a possible savings of 18,900 MH. If the cost of revising a document page is conservatively set at one MH, the savings over the 21,000 revised pages is still 12,600 MH. Applying these possible savings to the embedded revision in the three MH per page yields a modified page cost of 2.4 MH, implying a savings of 20 percent over the basic cost of the 91,000 non-revision sheets, or 54,600 MH (still using one MH per embedded revision). When considered together, the reductions amount to about 67,200 MH or 22 percent. Of course, the major benefit of this type of reduction is not that the job could be done more efficiently. Rather, a less ominous spectre of the documentation cost would allow the job to be performed in the first place.

The word processor would be the mainstay of smaller projects and those supplying only the lower levels of documentation. Indeed, for the large project generating a Part II Spec, the word processor can play a very large role, particularly in supporting the CPS guide, the programmer's notebooks, and the modeling and narrative sections of the computer program descriptions. Other areas are better served in other manners, especially the module design flowcharts, interface lists, data base structure and memory allocation, and the real-time cross reference. Of these, the most extensive and time-consuming in generation are the flowcharts and the interface lists.

Flowcharting an 'as-built' piece of software is prone to similar problems as writing the narrative text – adherence to drawing standards, typing comments into the flow elements, analysis of the code, etc. An alternative is to implement an automatic flowcharting tool. Without the specific approval of the procuring agency, however, this is proscribed by data item descriptions such as the DI-H-3277/M7. Why is there an aversion to auto-flowcharting? The usual auto-flow tool has been used to generate what amounts to an additional listing of the code which fails to enhance the software's supportability. The secondary goal of an auto-flow tool then should be to chart the design in an understandable and useful form. One of the more highly acclaimed design approaches is the top-down method, where the basic programming problem is iteratively divided into subsets of less general, more detailed problems that can eventually be easily solved on an individual basis. Flowcharting a top-down design in a likewise manner greatly enhances the usefulness of the documentation and its acceptability to the user.

BMAC has an auto-flow tool in the final stages of development, the Document Support System (DSS), which performs the flowchart generation in a top-down manner thus directly reflecting the levels of the design. At the same time, the source code image is separated into design levels corresponding to the generated flowcharts and formatted to reflect the logic nesting of each level. For illustration, a simple two-design level program has been generated and analyzed. Figure 2 shows the entire source image listing of the program which has been written for a Gould SEL 32/55 using a subset of S-FORTRAN developed by Caine, Farber & Gordon, Inc.

```
*         PART NUMBER
        PROGRAM X
        DATAPOOL  LBPPHUGD
        LOGICAL * 1  LBPPHUGD
C
C         1.0 THIS IS THE TOP LEVEL
C
C         ?  SHALL WE DO IT ?
C
          IF ( L )
C
C         1.1   DO  THE  FIRST  THING
C
          I = 1
C
C         1.2   DO  THE  SECOND  THING
C
C
C         1.2.1   FIRST  PART,  SECOND  THING
C
          K = 3
C
C         ? IS INDICATION NECESSARY ?
C
          IF ( I.EQ.2)
C
C         1.2.2  SET THE INDICATOR
C
          LBPPHUGD = .FALSE.
          END IF
          END IF
          END
```

Figure 2. Example Program

Separation of the code and attendant flowcharts per design level is depicted in Figures 3 through 7. The leading block of code, used for variable mnemonic attribute generation, common area references, and data initialization has been set aside in Figure 3. Figures 4 and 5 represent the top design level of 'executable' code. In Figure 4, code subsection 1.1 is shown with a replacement statement highlighted by dashed lines, while subsection 1.2 is represented only by a comment. This makes it immediately apparent that 1.1 is not subdivided into lower design levels, while 1.2 is. The flowchart of Figure 5 shows the logical flow of the top design level. Branching due to TRUE or FALSE states of the decision is indicated by 'T' or 'FF' within the logic paths leading from the decision block. Figures 6 and 7 show the second level of design (code subsection 1.2) pictured with the corresponding comment and process block displayed in the previous design level to aid in identification of the listing and flowchart. Those separated segments of code and flowchart lend themselves quite readily to integration with descriptive text for each of the design levels.

```
REV  :     STATEMENT
-----------------------------------------------
     :
 -   :    *         PART NUMBER
 -   :            PROGRAM X
 -   :            DATAPOOL   LBPPHUGD
 -   :            LOGICAL * 1   LBPPHUGD
 -   :    C
```

Figure 3. Leading Code Block

```
 REV  :   STATEMENT
--------------------------------------------------------
      :
  -   :  C 1.0 THIS IS THE TOP LEVEL
  -   :  C     ?  SHALL WE DO IT ?
  -   :        IF ( L )
  -   :  C     [   1.1   DO  THE  FIRST  THING
      :        [            +-----------------------
  -   :        [            !   I = 1
      :        [            +-----------------------
  -   :  C     [   1.2   DO  THE  SECOND  THING
  -   :        END IF
```

Figure 4. Top Design Level Code

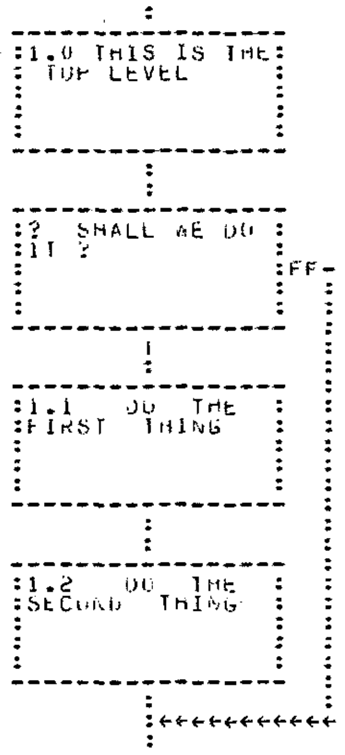

Figure 5. Top Design Level Flowchart

```
 REV  :   STATEMENT
-----------------------------------------------------------
      :
  -   :  C 1.2    DO  THE  SECOND  THING
  -   :  C      1.2.1   FIRST  PART,  SECOND  THING
      :                +-----------------------------------
  -   :                !   K = 3
      :                +-----------------------------------
  -   :  C      ? IS INDICATION NECESSARY ?
  -   :         IF ( I.EQ.2)
  -   :  C      [   1.2.2  SET THE INDICATOR
      :         [          +-------------------------------
  -   :         [          !   LBPPHUGD = .FALSE.
      :         [          +-------------------------------
  -   :         END IF
```

Figure 6. Second Design Level Code

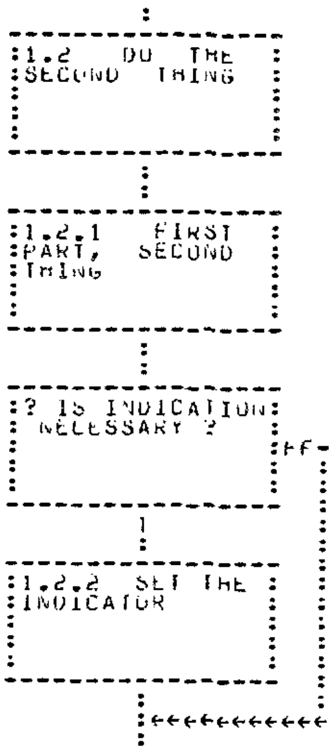

Figure 7. Second Design Level Flowchart

As evidenced, DSS can be used for analysis and documentation of existing code, provided certain coding standards have been followed. Additional applications are the generation of first-time flowcharts and the update of documentation per changes to design. Module designers generate a pseudo code of comments and generic logic structures, then use DSS to construct the design's flowchart. This gives excellent visibility to the design, its evolution, and its adherence to top-down and structured techniques. The example program of Figure 2 is in much the same format as would be generated by a designer under these circumstances. The two variables L and I have been inserted to provide proper syntax for the 'IF' logic structures, and would be replaced with proper mnemonics as they are defined.

An extremely optimistic expectation for the cost of the flowcharts and design level listings would be 1/4 MH per sheet. The amount of time required to generate all the sheets represented by Figures 3 through 7 was about 15 seconds, or 1/4-minute of real-time, leaving a great deal of spare computer time within the 15-second span. If we assume that text and 'boilerplate' comprise about one-half of each design document's non-listing pages, and the other half consists of flowcharts and design-level listings, we can arrive at a cost-savings estimate for combined word processing and auto-flowcharting. Applying the 22 percent word processing savings to one-half of the original 304,500 MH cost, or 152,250 MH, yields a savings of 33,830 MH. If we then assume the 15-second expenditure for generating the analysis of the example program applies to each sheet rather than all of them, then each sheet costs 15/3600 MH or 1/240 MH. Compared to our conservative 1/4 MH estimate for manual generation, this represents a savings of over 98 percent. The original cost of the 56,000 flowchart and design level sheets would be 14,000 MH at the 1/4 MH rate. A 98 percent reduction would be 13,720 MH for a total savings of 47,550 MH. Applied to a revised original cost of 166,250 MH (allowing 1/4 MH each for the flowcharts and design level lists assumed to comprise one-half of the 112,000 effective pages), the results are a savings of nearly 27 percent. Possibilities of this sort, especially given the conservative estimating, obviously start to bring high quality large documentation tasks into the realm of feasibility.

Documentation of module and system interfaces in a large system can be substantially more difficult than the documentation of the designs, given that many of the designs must be analyzed and correlated to give proper information for a few interfaces. This, of course, should then be accomplished in parallel with the progression of the design as the interfaces evolve. A prime candidate for support of this would be a Data Base Management System (DBMS) which correlates and maintains a data base of system interfaces. For this purpose, BMAC has instituted a DBMS with the MAXXIMUM data base manager by California Software Products, Inc., as its nucleus. This DBMS maintains interfaces which are implemented through the Gould SEL Datapool common memory facility and creates the mnemonic dictionaries used to link software load modules to the appropriate data spaces within the Datapool. In addition to the variable mnemonic, recorded data include variable attributes, dates of entry to the system and modification, logical location within the appropriate Datapool, software modules using the variable, and computers where using modules are resident. This obviously eases the pain of document generation as a great deal of this information can be gleaned from software module source code. Others, such as logical location within the Datapool, can be generated by the DBMS in response to variable attributes, unless constrained by operator input. An example of documented interface variables is given in Figure 8. Each of the two variable data entries gives the basic information stated above plus other pertinent facts such as the frequency at which the data is used and generated. Such listings can be easily generated for entire Datapool mnemonic dictionaries, or subsets thereof, to document tables and data bases according to the requirements contained within the Part II Spec.

The individual software module documentation is enhanced by the DBMS-produced module interface listing, shown in part in Figure 9. The module in question is shown boxed in asterisks on the left with individual interface linkages drawn to related modules shown boxed on the right portion of the figure. Each link shows the variable mnemonic for the data space that it represents, the type of data space (i.e., integer byte, logical byte, etc.), the data space array size, and the relative flow of data depicted by an appropriately oriented caret.

Interface documentation on the part of the DBMS is top-down in

```
                          KC-135 NAVIGATOR STATION PRIVATE MEMORY CPU 04 (QTP 14 / 03/06/81)
IBPMFZRQ  MASTER FREEZE REQUEST
          -UNITS-      -RANGE-       -ACCURACY-      -RESOLUTION-       -TYPE-      (REF PAGE K0999)DATE CHANGED 12/31/1980
          N/A          0-255         N/A             +-1                1 INTEGER BYTE
          DATE ENTERED 24 MAY 1978
                       -MNEMONIC-    -MODULE NAME-           -PART NUMBER-        -FREQ- -MODE- -CPU- -DATE ENTERED- -DATE CHANGED-
          GENERATED BY  RNIBSEPS     OPSYS/CPU 4 ST EX PC    291-47104-1      F   20 HZ  ALL     4      05/25/1978      02/19/1980
          USED BY       RWIBPRS5     PROCESS SIMULATOR ST    291-40230-4      A   20 HZ  BOTH    4      02/25/1981      12/31/1980
                        RNIBSEPS     OPSYS/CPU 4 ST EX PC    291-47104-1      F   20 HZ  ALL     4      05/25/1978      02/19/1980
          DETAIL DESCRIPTION :    (BASE801)

IBPPOSPT  POINTER TO THE ARRAY POSITION OF    ELAPSED FRAME TIME; VALUE IS RELATED TO CPU   DATE CHANGED 12/31/1980
          -UNITS-      -RANGE-       -ACCURACY-      -RESOLUTION-       -TYPE-      (REF PAGE 00999)
          TBD          1-7           TBD             TBD                1 INTEGER BYTE            INITIALIZE TO ZERO
          DATE ENTERED 26 SEP 1977      CPU INITIALIZATION REQUIRED
                       -MNEMONIC-    -MODULE NAME-           -PART NUMBER-        -FREQ- -MODE- -CPU- -DATE ENTERED- -DATE CHANGED-
          GENERATED BY  CPU INIT     CPU 05 INIT DATA        291-47100-9      -   N/A    N/A     5      12/31/1980      02/24/1980
                        CPU INIT     CPU 04 INIT DATA        291-47100-8      M   N/A    N/A     4      12/31/1980      02/24/1980
          USED BY       MWOBINIT     INITIALIZE MAINT. EX    291-40726-2      -   N/A    ALL     5      03/03/1980      04/01/1980
                        MWIBEXEC     MAINTANENCE EXEC        291-40726-1      D   20 HZ  ALL     5      11/06/1978      02/27/1981
                        RWIBPEXX     OPSYS/CPU EX NUC PEC    291-40229-1      E   20 HZ  ALL     4      05/25/1978      03/13/1980
                        CPU INIT     CPU 05 INIT DATA        291-47100-9      -   N/A    N/A     5      12/31/1980      02/24/1980
                        SWOLIS       INSTRUCTOR STA CONT     SWOLIS           J   BKGRD  ALL     5      02/14/1978      02/24/1980
                        CPU INIT     CPU 04 INIT DATA        291-47100-8      M   N/A    N/A     4      12/31/1980      02/24/1980
                        RWIBPE05     OPSYS/CPU EX NUC PEC    RWIBPE05         C   20 HZ  BOTH    5      08/26/1977      02/02/1980
          DETAIL DESCRIPTION :    (BASE803)
```

Figure 8. Interface Data Base Documentation

```
******************                                                                  ******************
*                *                                                                  *                *
* RL3RO2RG KF  3 *                                                                  * LNKR     KF  0 *
* 291-48402-1    *-------*                                                  *-------* REDIFON        *
* REVISION:  F   *       !                                                  !       * REVISION:      *
*                *       !                                                  !       *                *
******************       !                                                  !       ******************
                         !                                                  !
                         !<-S<-S<-S<-S<-S<-S  IRSXO2SB  <-S<-S<-S<-S(IB   1)<-!
                         !<-S<-S<-S<-S<-S<-S  IRSXO2SC  <-S<-S<-S<-S(IB   1)<-!
                         !<-S<-S<-S<-S<-S<-S  IRSXO2SN  <-S<-S<-S<-S(IB   1)<-!
                         !<-S<-S<-S<-S<-S<-S  LRSDILSB  <-S<-S<-S<-S(LB   1)<-!
                         !<-S<-S<-S<-S<-S<-S  LRSDILSN  <-S<-S<-S<-S(LB   1)<-!
                         !<-S<-S<-S<-S<-S<-S  LRSEMERB  <-S<-S<-S<-S(LB   1)<-!
                         !<-S<-S<-S<-S<-S<-S  LRSEMERN  <-S<-S<-S<-S(LB   1)<-!
                         !>-S>-S>-S>-S>-S>-S  RWSPOPF2  >-S>-S>-S>-S(EW   1)->!
                         !
```

Figure 9. Inter-Module Interfaces

essence, starting with the contents of the data base and moving down to the module interface level. On the other hand, DSS has an interface documentation capability that starts with the module internal interfaces and moves up to the system level. An interface analysis of the brief example program is shown in Figure 10. The variable mnemonics I, K and L are shown to be unique to the module, while LBPPHUGD has been found in a predesignated common memory data base, thus allowing DSS to obtain array-size information and the variable's description. Mnemonic 'I' is shown to be an output of level 1.0, by virtue of an 'O' in the left column of the I/O matrix in the right portion of the figure. The right-hand column (for level 1.2) has no entry, indicating the 'I' is not used on the second design level. DSS has obviously shown us that the module is suspect, since a unique variable is generated as an output in one design level, but is never used as an input. Similarly, mnemonics are shown to be inputs by the presence of an 'I' in the design level column of the matrix, and mnemonics both used and generated are represented with a 'B'. As DSS analyzes modules, external interface information may be *retained and later used to document interfaces between modules* within a system, and between different systems of multiple modules. This difference in approach between the DBMS and DSS is summarized in Figure 11. The redundant aspect of system interface documentation makes an excellent cross-check mechanism, and helps to ensure data base and interface integrity.

```
                                                         1  1
                                                         0  2
                                                         0  0
                                                         0  0
                                                         0  0
                                                         0  0
                                                         0  0
                                                         0  0
                                                         0  0
                                                         0  0

I         (****)  **** LOCAL VARIABLE          ****      O  I
K         (****)  **** LOCAL VARIABLE          ****      U  O
L         (****)  **** LOCAL VARIABLE          ****      I
LBPPHUGD  (001 )  PHUGOID FLAG                           O  O
```

Figure 10. Example Program Interfaces

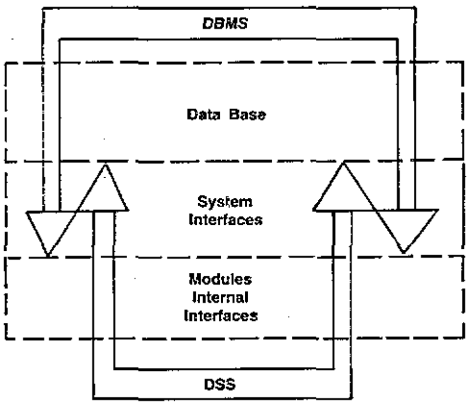

Figure 11. DBMS/DSS Perspectives

As with DSS, the DBMS process has the added benefit that discrepancies can almost always be detected and brought to light as soon as the offending source code is analyzed and its interface expectations are compared to the data base. the DBMS produces an output that documents these discrepancies, as depicted in Figure 12. This particular example indicates that the module RO2BBAOU has defined several mnemonics to be in a common memory area, but never references these mnemonics in executable code. Additionally, the mnemonics are shown not to exist within the data base. This has probably arisen from a previous deletion of the mnemonics from the data base and executable code, while a somewhat careless change implementation has allowed the common definitions to remain. Two other variables are shown to disagree in array sizes between those expected by the module and those implemented by the data base.

While cost estimates for the preparation of interface documentation are not really available, it should be apparent that the process is certainly no easier than manual design documenting. Suffice it to say *that the process which generated the information, of which Figure 9 is* a part, took less than four minutes of computer real-time. The possibilities of savings should speak for themselves.

In order to efficiently perform these flowcharting and interface documenting capabilities, some standards and conventions should be imposed upon the software to be documented. In the instance of the interface DBMS, variable mnemonics are required to utilize the first three characters to show variable attributes of type and size, and whether they are to reside in a particular region of the Datapool common area or are unique to the module. DSS requires that the code follow structured precepts and be designed in a top-down fashion. Submodules and design levels within the code must then be commented with a numbering system that indicates to which submodule and design level a segment of code belongs. Figure 13 illustrates the hierarchical numbering of the example program processed by DSS. It can be successfully argued that other benefits of such practices (e.g., increased reliability) far outweigh the trouble of standards implementation, without even considering the applications *to documentation.*

The basic requirements of a general Part II Spec can be satisfied with the word processing, auto-flowcharting, and auto-interface-documenting described above. Other items such as the Real-Time Cross Reference can be completed with an extension of the DBMS source code analysis. Once these or similar mechanisms are implemented, however, there is still room for improvement. Correlation of documentation is a consideration, as are consolidation of document integration and generation of such items as test discrepancy reports, change requests, etc. Document correlation may well take the form of a DBMS function that maintains data on functional requirements documents, implementation requirements documents, and design and test documents. A change to a functional requirement paragraph could then be linked to subordinate implementation requirements and design and test documents, generating a 'need to change' list. Likewise, a module design change could be picked up by the DBMS and result in an output indicating a need to change the appropriate text through word processing.

```
*** MODULE R02RBAUU  291-43172-1    K  06/26/1982 EMPLOYES 110 DATAPOOL VARIABLES IN THE FOLLOWING
CPUS -    9
THIS REVISION ( K) HAS PREVIOUSLY BEEN VERIFIED
VARIABLE IBPAOU10 SHOWS UP IN COMMON BLOCK / EXTENDED MEMORY OF PROGRAM BUT IS NEVER USED BY PROGRAM
VARIABLE IBPAOU10 DOES NOT APPEAR IN DATA BASE
VARIABLE IBPAOU20 SHOWS UP IN COMMON BLOCK / EXTENDED MEMORY OF PROGRAM BUT IS NEVER USED BY PROGRAM
VARIABLE IBPAOU20 DOES NOT APPEAR IN DATA BASE
VARIABLE IBPAOUCD SHOWS UP IN COMMON BLOCK / EXTENDED MEMORY OF PROGRAM BUT IS NEVER USED BY PROGRAM
VARIABLE IBPAOUCD DOES NOT APPEAR IN DATA BASE
VARIABLE IBSAOUPR SHOWS UP IN COMMON BLOCK / EXTENDED MEMORY OF PROGRAM BUT IS NEVER USED BY PROGRAM
VARIABLE IBSAOUPR DOES NOT APPEAR IN DATA BASE
VARIABLE IBSATGTI IN STATION BO - ARRAY MISMATCH (MODULE=    1) (DATA BASE=  200)
VARIABLE LBPPCDPA IN STATION BO - ARRAY MISMATCH (MODULE=    3) (DATA BASE=    5)
```

Figure 12. Interface Discrepancies

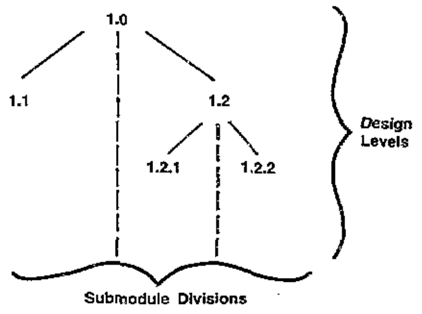

Figure 13. Example Program Design Levels and Submodules

Document integration could entail consolidation of the output of these various processes in order to avoid the manual sorting and shuffling of papers. Use of laser printers, plotters, and the like, if available, is not out of the question. With the obvious savings involved, the automatic processes explored can make giant strides toward high-quality, reasonably priced documentation.

REFERENCES

1. Boehm, B. W., "Software Engineering Economics," Prentice Hall, 1981, p. 574.

2. ibid, p. 572.

3. ibid, p. 574.

ACKNOWLEDGEMENTS

Special thanks to Stan Chilcott and Jim Richardson of Boeing Military Airplane Company, whose programming expertise was indispensable.

ABOUT THE AUTHOR

Mr. Kerry M. Atchinson received his BSEE from Oklahoma State University in 1977. He has six years experience at Boeing Military Airplane Company where he is currently a Simulation Software Engineer in the Military Training Systems organization. His main responsibilities are the development of organizational standards and guidelines for training systems software, and development of training system supervisory and support software.