TITLE  SYSTEM INFORMATION DATABASE:
AN AUTOMATED MAINTENANCE AID

AUTHORS:  BRICE, LINDA
CONNELL, JOHN

SUBMITTED TO 1984 NATIONAL COMPUTER CONFERENCE
LAS VEGAS, NEV., JULY 1984

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Los Alamos National Laboratory
Los Alamos, New Mexico 87545
by LINDA BRICE
and JOHN CONNELL
Los Alamos National Laboratory
Los Alamos, New Mexico
ABSTRACT

Documenting of application systems has long been considered a necessary evil. Necessary because documentation provides a map to present systems, serves as a maintenance aid and is required by the auditors; evil because it is an activity generally dreaded by those who develop the systems. Since normal behavior surrounding unpleasant chores is avoidance, application systems documentation is sometimes absent and often incomplete.

Documenting may be unpopular for a number of reasons, including psychological ones. One very obvious problem is that, except for a few automated tools at the program level, documenting is a manual process used in a high-technology environment. Automating the process is a way to reduce the laboriousness of portions of the task.

This paper is a case study of how one DP organization applied student labor and a relational DBMS in a prototyping environment to automate much of their applications systems documentation function. The capabilities, fringe benefits and future enhancements of the tool are discussed.

KEYWORDS: Documentation; Tools; Software Maintenance; Rapid Prototyping; Administrative Data Processing; Dictionaries
INTRODUCTION

Why should maintenance aids be automated? In many installations system documentation is still a cumbersome manual process. There are automated data dictionaries and program documentors on the market, but few link to other aspects of an organization's functions, and most take several years to populate with data. Some organizations commit to five or ten years worth of data gathering and data entry, unassured of the end results. Others accept as a fact of life that manual documentation is not an effective maintenance aid, but continue setting up a framework with strict requirements and standards.

This paper relates how a relational data base management system was used to develop an in-house automated documentation system for the Administrative Data Processing Division of the Los Alamos National Laboratory. The database has been given the acronym SID, System Information Database. It contains much of the documentation pertaining to production application systems. This documentation has historically been maintained manually in Central File folders. At the time of this writing, SID has proven to be very effective for entering, updating, and retrieving documentation data rapidly and accurately.
WHY THE NEED TO DOCUMENT

Documentation is considered to be a map of present systems and a valuable aid to maintenance programmers. Accurate documentation is also a reliable guide to relationships within and between systems. It provides a means for reducing the risk of introducing errors during maintenance work. If an error does occur, a visual picture of control flow is available to help locate the source of the error. In the normal course of events, clear documentation makes staff turnover less disruptive by providing a useful training aid. Finally, adequate documentation will satisfy auditors' requirements for information about how systems work.

Data processing has long been admonished by the literature to document in certain standard ways. Most shops were led to believe, by the literature of the 1970's, that Volume, Table of Contents (VTOC's), IBM's Hierarchical Input Process Output (HIPO's), and flowcharts, for example, were the best tools for documentation and were necessary. Now, we are told to produce Data Flow Diagrams, Structure Charts, Chapin Charts, data models, Jackson diagrams, Warnier-Orr diagrams plus myriad forms supplied by structured methodologies.
Many installations simply have not sorted out which old tools to discard, which new ones to adopt, what to make retroactive, or whether all tools need be applied at the system, task and program level or not. Most organizations have viewed documentation as a program-level activity, with recent emphasis on the data element-level. There is much more than programs in the makeup of most application systems. They are also comprised of operating system level procedures, data base interfaces, data files and other elements. Documentation must not only be present, it must be flexible. Few DP organizations can bear the expense of throwing a system away and rewriting it from scratch. When "... the intent is to modify functionality or capability or even performance, the trend is to add code, a front end, or a box. ... 'Add on. not replace' is the trend in software."[1] Documentation must be easily enhanced, just like software. Martin and McClure state, "... what is needed is succinct, high-quality documentation that is easily accessible and easily updatable. To be maintainable, programs and their associated documentation must be flexible and extensible."[2] To that statement we could add that all documentation pertaining to an applications system must fit the same description as that for a program.
BASIC ELEMENTS OF DOCUMENTATION GENERALLY NEEDED FOR EACH APPLICATION

Whatever tool is applied at whatever level, the basic elements of documentation needed for a typical business application include:

- the basic purpose of the system;
- identification of the customer;
- how the system runs (tasks, procedures, call files, jobs, operating system commands);
- how execution begins and proceeds;
- what groups of higher level language or fourth generation language instructions exist;
- how the groups of language (or programs) get invoked;
- what functions are performed;
- what files exist;
o how is the data processed and by which tasks or programs;

o what does the output (input) look like (files, screens, reports, etc);

o who is responsible for the system maintenance.

Whatever the size of the hardware, the size of the application, the programming language employed, the number of staff members, the use of a database management system or not, these types of basic elements need to exist for maintainers and auditors of the system.

WHY DOCUMENTING IS SO UNPOPULAR

Documentation, useful if not absolutely necessary, is the least favorite part of most DP professionals' job duties. This is so because it is seldom scheduled. When schedules slip, system implementation is the more important feature. There must be a system. The documentation portion of the schedule, often inadequately allotted at the start, is diminished because it is often performed after the fact and because it is usually a clumsy, manual system. Sometimes documentation begins when maintenance begins.[3]

-6-
Documentation in ADP was completely manual prior to the development of SID and included:

-- A Volume Table of Contents (VTOC) describing the hierarchy of tasks. A manually drawn set of boxes within a strict format. The major functions of the system appear as text within the boxes of this system schematic (Figure 1). The VTOC was initiated during system design and maintained during the life of the system. It was normally produced after system implementation, to merely fulfill a documentation requirement, and was often not maintained because of the necessity to manually redraw and retype the chart.

-- A Hierarchical Input Process Output describing the flow of input/output with respect to the functions of a program or task. Special symbols to represent files, output listings, direction of flow (arrows) were manually drawn with the aid of a template and the narrative typed (Figure 2). HIPO's were intended to be a design aid, but were usually produced post-implementation and then only due to standards requirements. Obviously, due to the nature of the format, changes of any consequence required redrawing of one or more pages, or a manual cut-and-paste procedure. Such inconvenience discouraged the maintenance of the charts to accurately reflect the state of the system as it changed character over time due to maintenance and enhancements.
-- Indices of programs and files providing simple lists, usually alphabetized. Other information, such as what task invoked the listed program, or what files are referenced by the program was usually present (although some of the data existed in other forms in the file). The frustration in manually maintaining such lists is that the data must be recorded at least twice (on the program index, the I/O files are listed; on the file index, the referencing programs are listed).

-- File and data element information. Data elements were typically described by a record layout form (Figure 3). The record layouts were often hand-drawn.

-- Program listings. Program listings were maintained in hard copy form in folders arranged in an order meaningful to the organization (by section, by function, etc). The listings were checked out to maintenance parties in a library-type arrangement.

AUTOMATION CAN MAKE DOCUMENTATION MORE PALATABLE

Streamlining of documentation procedures may improve the product to the point that it becomes a true maintenance aid instead of a mere fulfillment of standards requirements. There are psychological reasons programmers are more
comfortable with automated tools than with manual ones. DP, like the shoemaker with his barefoot children, automates the lives of others, but often has no time to automate its own business. Programmers are naturally frustrated when forced to deal with internal paper work when they are accustomed to automation in every other aspect of their lives.

If manual processes are clumsy, they tend to produce incomplete and inaccurate results. While management makes rules in the form of standards, having an understandable profit incentive, they reinforce the message to their staff that the most important part of their job is to get the system up and running. Of course, the message is well received by programmers, who often view documentation as a nuisance.

Automation of documentation has all of the advantages of any other automated system, including interactive retrievals, simultaneous access by several parties and easy aggregates. One particular advantage of automation of documentation is the retrieval of information across systems. For example, the manual documentation shows program and file relationships within a particular system, but if one wanted to list every program that reads file xxx because the format must change to increase the field length of a data item, then all manual documentation for systems suspected to relate to the
file must be searched, or all machine readable files across those systems must be searched to complete the list; an easy retrieval for a properly formatted system information database. Size considerations, an aid in estimation of the effort required for a job, are also available e.g., the number of files within the number of systems that reference Purchase Order Number or one of its aliases. "The most common error in documentation is to provide masses of detail ... but little on overall organization ... and on the relationship between parts."[4]

AUTOMATING DOCUMENTATION: A CASE STUDY

At Los Alamos National Laboratory, it was agreed upon by management and staff that an automated documentation process should be attempted. A relational database system was already licensed in-house, had proved to be an excellent tool for other applications and was chosen to inventory and manage parts of our documentation function. There existed, however, a resource problem. All available analysts, designers and programmers were committed to other projects. Given the work load facing the entire division, there was little justification in hiring staff for the documentation project, which was considered overhead. It was not development of applications desired by the customers who pay the bills. There was also a little skepticism on the part
of management. There had been no official cost/benefit study performed for the project and management could not be certain it would be worth the effort to disturb the status quo to implement a new documentation system when the staff was in the throes of a great deal of new development.

By a fortunate circumstance, the Administrative Data Processing Division at Los Alamos was host to four young men from the service academies for the summer.* The Service Academy Research Associates (SARA's) were from the Air Force Academy and the Naval academy; three of them were in their senior year; one was a computer science major; none had practical data processing experience. They were enthusiastic about learning a state-of-the-art tool, so it was decided to assign them the documentation project, even though they could not work as a true team since their four to six-week tenures overlapped very little. Armed with a name, the System Information Database (SID), and a database management tool, they produced a prototype that proved to be quite successful in convincing management and staff that the documentation procedures could indeed change for the better.

While the first SARA was en route to Los Alamos, a Systems Requirements Definition was produced as a guide to the current manual system and what was hoped to be accomplished with SID. Normally, a Systems Design Document follows the Requirements Definition in the development of any new project. In this case, however, the detailed design was replaced with the prototype version of the system.

A pilot system was rapidly available for management to evaluate in terms of cost/benefit and for the staff to evaluate in terms of usability. The pilot project had small scale actual data. Some data was entered for small but complete systems.

The system was refined by submitting the prototype version to selected members of the programming staff for critique. Tables were easily restructured to add and delete data elements or to modify attributes, without the loss or troublesome reloading of any of the real data. Additional live data was loaded from a hierarchical database on a separate computer via magnetic tape. Live data was also loaded from files which programmers had set up to keep track of various systems for which they were responsible. It was interesting to note that many programmers had already discovered that the manually maintained central files were inadequate for maintenance purposes and that several members...
of the staff had taken steps to record applications data in a more usable state.

A recent survey of programmer opinion indicated that the current ADP staff was 100% in favor of maintaining an automated system to map the state of present systems and the evolution of future systems. When a representative task force of the programming staff viewed demonstrations of the retrievals, they responded favorably.

Some of the automated retrievals which replaced manual documentation elements include: Volume Table of Contents (Figure 4), Hierarchical Input Process Output (Figure 5), Index of Programs, Index of Files, Index of Tasks, Catalog of Systems (Figure 6). The Volume Table of Contents is somewhat different in format from the original. To allow for an unrestricted number of high level functions, the information is spread down the page instead of across. The informational elements are retained, however, and both hierarchical and sequencing attributes are preserved. A Catalog of Systems relates files to programs, programs to tasks and tasks to systems. In the example in Figure 6, the Capital Equipment Budget System (CEBS) is documented. CEBS is identified as system 23. Task 2301 is a procedure file which executes three programs - 230601, 230605 and 230625. Each program is also identified by its generic name. Files
appearing as I/O within the programs are documented in the rightmost column. Source data is input to the data base using the input screen tools supplied by the database management system (Figure 7). Updates to documentation of the present system are accomplished using the same screens.

FRINGE BENEFITS

The database was brought up with the intent of aiding programmers map present and future systems. However, once in place, it proved to provide several side benefits. A matrix describing system identifiers and associated responsible programmers had been maintained on word processing equipment. A similar matrix detailing application system, organizational section where the functional responsibility for that application resides, and programmers identified in order of level of responsibility (primary responsibility, backup to primary responsibility and secondary backup responsibility) is now achievable by a fairly simple merge of relations. The query language commands are collected into an executable procedure so that the matrix can be produced with one operating system level command. The word processing files have been deleted and the clerical staff updates employee information as it relates to system responsibility directly on the database. Section leaders (first line management to whom the
responsible programmers report) likewise update responsibility changes directly on the database. Figure 8 is an example of the Responsibility Matrix. Of course, responsibility can be retrieved by staff member as well as by application system. It is sometimes useful to management to know by employee, to which systems that employee maintains responsibility, and what constitutes the level of responsibility. Once system responsibility data is captured, it is a simple step to report organizational entity, telephone number and physical location for the staff, either as a complete organizational report or as ad hoc retrievals for single individuals or groups of individuals.

Another fringe benefit of storing gross system data in one place is the ability to estimate system size. Many installations can name the modules present in a system, but few know much about actual system size as expansion/contraction is continually taking place with modifications. There is an occasional need to give at least approximate figures answers to questions like "How long will it take us to convert completely to a new hardware vendor?" or, "What is the estimate for converting to a new language version or a different control language?". Those questions are frequently not just academic since entire installations have been known to undergo hardware vendor changes and it is not
unusual for vendors of software to cease support of past versions. Approximate figures for lines of code per language, languages per system, programs per system, tasks (operating procedure level commands) per system and other sums can provide the basis for estimating conversion effort and, therefore, monetary cost. Such queries can be processed easily by the count and sum features of most databases.

FUTURE ENHANCEMENTS

While the primary intent of the database is to serve the current and future programming staff who maintain present systems and develop new systems, the functions can be expanded to include the operations side of systems production. Run/recovery instructions, file access and permits, account restrictions, job setups, file retentions, expected outputs and other operations data can be appended to system, task, program, file or data element relations as appropriate. Operations information is a natural addition as operators and production controllers are also interested in employee system responsibilities and system functional descriptions, which have already been described in the database.

Information concerning system functions, responsibilities and operations forms a useful link to the controlling of
resources and the measuring of activities associated with a system. The level of activity against a system is a guide to future staffing in an organization. Activity in the form of customer requests for service (maintenance, enhancements) on a particular system can be married to the System Information Database to get a complete picture of current system activity levels. For example, it can be noted that System #98 is General Ledger, that task #107 Account Update executes 12 programs and 7 files (from SID), that the task is executed approximately 30 times per month (from SID), that program #203 aborted seven times last month (from SID with operations data), and that program #203 had five Service Requests logged against it in the last six weeks (from the Resource Control or Metrics database). Other data, such as the effort required to complete the requests for service on the program, and past history of the program can be used in assessing staffing levels for the system as well as considerations for a redesign of the program.

CONCLUDING REMARKS

No database, even a modern relational database, is magic. The organization considering support of a System Information Database must commit to some amount of overhead. As in the case of the automated systems we deliver to our customers, data must be entered, the database tool must be understood
and more likely than not, programs will have to be designed and maintained to perform sophisticated retrievals and to provide links from one database to others.

When SID was developed in ADP at Los Alamos, the prototype was brought up almost entirely by the SARAs, a real tribute to the ease of use of the relational data base management system. Yet, several programs were required, adding to the overhead of maintenance and documentation for those remaining after the student apprentices have left. Like all systems, data processing's management information systems must be staffed to watch for and prevent system degradation.

REFERENCES


FIGURE CAPTIONS:

Figure 1. Visual Table of Contents (VTOC)
Figure 2. Hierarchical Input Process Output (HIPO)
Figure 3. Record Layout
Figure 4. VTOC
Figure 5. HIPO
Figure 6. System Catalog
Figure 7. SID Data Entry Screen
Figure 8. SID Retrieval
BATCH PROCESS NO 12

- DAILY UPDATE
- BACKUP RECOVERY
- MONTH END REPORTING
  - EDIT AND UPDATE
  - PRODUCE MASTER LIST
  - PRODUCE MASTER LIST
BATCH PROCESS NO. 27

1. EDIT TRANSACTIONS
2. UPDATE SEQUENTIAL FILE A
3. PRODUCE UPDATE REPORT

PROCEDURE 2
PRODUCE MASTER LIST

PROCEDURE 3
1. PURGE OLD DATA BASE
2. CREATE NEW DATA BASE FROM SEQUENTIAL FILE

FILE A
UPDATE REPORT
MASTER LIST
NEW DB
<table>
<thead>
<tr>
<th>FIELD NAME</th>
<th>EMPLOYEE NUMBER</th>
<th>EMPLOYEE NAME</th>
<th>DATE OF BIRTH</th>
<th>SEX</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHARACTERISTICS</td>
<td>X(6)</td>
<td>X(14)</td>
<td>X(6)</td>
<td>9</td>
<td>...</td>
</tr>
<tr>
<td>RELATIVE POSITION</td>
<td>1–6</td>
<td>7–20</td>
<td>21–26</td>
<td>2?</td>
<td>...</td>
</tr>
<tr>
<td>INPUT FILES</td>
<td>PROGRAM</td>
<td>OUTPUT FILES</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>------------</td>
<td>--------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRANSACTIONS</td>
<td>230501</td>
<td>FILEA REPORT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FILE A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EDIT TRANSACTIONS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>UPDATE FILE A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PRODUCE UPDATE REPORT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FILE A</td>
<td>230605</td>
<td>MASTER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PRODUCE MASTER LIST</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FILE A</td>
<td>230625</td>
<td>NEW DB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OL8 DB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PURGE OLD DB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CREATE NEW DB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSTEM ID</td>
<td>TASK (PROCEDURE) ID</td>
<td>PROGRAM NAME</td>
<td>FILE(S) USED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>---------------------</td>
<td>--------------</td>
<td>--------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23 CEBS</td>
<td>2301</td>
<td>230601</td>
<td>FILE A REPORT</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>230605 MASTER LIST</td>
<td>FILE A MASTER</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>230625 NEW DB</td>
<td>FILEA NEW DB</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>OLD DB</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NEW DB</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## RESPONSIBILITY TABLE

<table>
<thead>
<tr>
<th>SYSTEM ID</th>
<th>SYSTEM NAME</th>
<th>ORGANIZATIONAL SECTION</th>
<th>PRIMARY</th>
<th>1st BACKUP</th>
<th>2nd BACKUP</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>PAYROLL</td>
<td>EMPLOYEE INFORMATION</td>
<td>HAWKINS</td>
<td>RICH McCALISTER</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>COMMITMENTS</td>
<td>ACCOUNTING AND OPERATIONS</td>
<td>TOMLINSON</td>
<td>HUDGINS ARMSTRONG</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>CAPITAL</td>
<td>BUDGLT AND PLANNING</td>
<td>ROYBAL</td>
<td>HILL</td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>GENERAL LEDGER</td>
<td>ACCOUNTING AND OPERATIONS</td>
<td>HUDGINS</td>
<td>OSBORN</td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>PROPERTY</td>
<td>MATERIALS</td>
<td>ARMSTRONG</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...