System information database:  
An automated maintenance aid

by LINDA BRICE  
and JOHN CONNELL  

Los Alamos National Laboratory  
Los Alamos, New Mexico

ABSTRACT

Documenting 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 regarding 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, documentation is a manual process used in an automated environment. Automating the process is a way to reduce the laboriousness of the task.

This paper is a case study of how one data processing organization applied student labor and a relational database management system in a prototype to automate much of their applications systems documentation function. The capabilities, fringe benefits, and future enhancements of the tool are discussed.
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, unsure of the results. Others accept as a fact of life that manual documentation is not an effective maintenance aid, but continue to set up frameworks with strict requirements and standards.

This paper shows how a relational data base management system was used to develop an in-house automated documentation system for the Administrative Data Processing (ADP) 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 the “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 professionals have long been admonished to document in certain standard ways. Most shops were led to believe, by the literature of the 1970s, that visual tables of contents (VTOCs), IBM's hierarchical input process output (HIPO), and flow charts, 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, and Warnier–Orr diagrams, as well as 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 or not all tools need to be applied at the system, task, and program level. Most organizations have viewed documentation as a program level activity, with recent emphasis on the data element level. There is much more than a program in the makeup of most application systems. They are also composed of operating system procedures, database 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.” Documentation must be enhanced easily, just like software. Martin and McClure state that “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.” 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

Regardless of the tool used or the level at which it is applied, the basic elements of documentation needed for a typical business application include:

1. The basic purpose of the system
2. Identification of the customer
3. How the system runs (tasks, procedures, call files, jobs, operating system commands)
4. How execution begins and proceeds
5. Which groups of higher level languages or fourth-generation language instructions exist
6. How the groups of languages (or programs) are invoked
7. Which functions are performed
8. Which files exist
9. How is the data processed—and by which tasks or programs
10. What the output (input) looks like (files, screens, reports, etc)
11. Who is responsible for the system maintenance

Whatever the capacity of the hardware, the size of the application, the programming language employed, the number of staff members, or whether a database management system is used 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 often the least favorite part of most DP professionals' duties. This is so because documentation is seldom scheduled as part of the job. When schedules slip, system implementation is a 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

Documentation in ADP was completely manual prior to the development of SID and included several elements: First was a visual table of contents (VTOC) describing the hierarchy of tasks. This is 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 often was not maintained because of the necessity to manually redraw and retype the chart.

The next item was a hierarchical input process output (HIPO) describing the flow of input and output with respect to the functions of a program or task. Special symbols to represent files, output listings, and direction of flow (arrows) were drawn by hand with the aid of a template, and a narrative was typed (Figure 2). HIPOs were intended to be design aids, but were usually produced post-implementation and then only because of 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 because of maintenance and enhancement.

Next were the indices of programs and files, which provided simple lists, usually alphabetized. Other information, such as what task invoked the listed program, or what files were referenced by the program was usually included (although some of the data existed in other forms in the HIPO). The frustration in manually maintaining such lists is that the data must be recorded at least twice (the I/O files are listed on the program index; the referencing programs are listed on the file index).

Also included was information about file and data elements. Data elements were typically described by a record layout form (Figure 3). The record layouts often were hand-drawn.

Finally, there were program listings, which were maintained in hard-copy form in folders arranged in an order meaningful to the organization (by section, by function, and so on). The listings were checked out to maintenance programmers 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 that programmers are more comfortable with automated tools than with manual ones. Data processing professionals, like the shoemaker with his barefoot children, automate the lives of others, but often have no time to automate their own business. Naturally, programmers become frustrated at being forced to deal with internal paper work when they are accustomed to automation in every other aspect of their work.

If manual processes are clumsy, they also tend to produce incomplete and inaccurate results. Although management makes rules in the form of standards, having an understandable incentive for profit, they reinforce the message to their staff that the most important part of a 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.

Automated 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 automated documentation is the retrieval of information across systems. For example, 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. As Brown writes, “the most common error in documentation is to provide masses of detail... but little on overall organization... and on the relationship between parts.”

AUTOMATING DOCUMENTATION: A CASE STUDY

At Los Alamos National Laboratory, management and staff agreed 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 a development of an application desired by the customers who pay the bills. There was 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 ADP Division was host for the summer to four young men from the service academies.* The Service Academy Research Associates (SARAs) came to us from the Air Force and the Naval academies; three of them were in their senior year, one was a computer science major, and 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, 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 we wanted to accomplish 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 and benefit and for the staff to evaluate in terms of usability. The pilot project had small-scale actual data; data were 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 were loaded from a hierarchical database on a separate computer via magnetic tape. Live data also were loaded from files that 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 that replaced manual documentation elements include the VTOC (Figure 4), HIPO (Figure 5), index of programs, index of files, index of tasks, and catalog of systems (Figure 6). The VTOC 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 that 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 database 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

SID was devised with the intent of helping programmers to map present and future systems. However, once in place, it provided several other 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 by level of responsibility (primary responsibility, back-up to primary responsibility, and secondary back-up responsibility) can now be made 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 record responsibility changes directly on the database. Figure 8 is an example of the responsibility matrix. Of course, responsibility information can be retrieved by name of staff member as well as by application system. It is sometimes useful for management to know—by employee—for which systems each employee maintains responsibility, and what constitutes the level of responsibility. Once system responsibility data are captured, it is a simple step to report organizational entity, telephone number, and location for members of staff, either as a complete organizational report or as 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 list the modules present in a system, but few can report much about actual system size, because expansion and contraction take place continuously with modification. There is an occasional need to give at least approximate-figure answers to questions about how long it will take to convert completely to a new hardware vendor or what the estimate is for converting to a new language version or a different control language. These questions frequently are not just academic; entire installations can change hardware vendors, and it is not unusual for vendors of software to cease support of earlier 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 programming staff who maintain present systems and develop new ones, the functions can be expanded to include the operations side of systems production. Run and recovery instruc-
tions, 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 because operators and production controllers are also interested in employee system responsibilities and system functional descriptions, which have already been described in the database.

Information about system functions, responsibilities, and operations can form a useful link to controlling resources and measuring 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 past 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 history of the program, can be used in assessing staffing levels for the system as well as for considerations in the program's redesign.

CONCLUDING REMARKS

No database, even a modern relational database, is magic. The organization considering support of a SID 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 another.

When SID was developed by 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 database 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