DIFFERENCES BETWEEN BUILDING A TRADITIONAL DSS AND AN ODSS: LESSONS FROM THE AIR FORCE'S ENLISTED FORCE MANAGEMENT SYSTEM

Warren E. Walker

August 1989

P-7584

The RAND Corporation, 1700 Main Street, P.O. Box 2138, Santa Monica, CA 90406-2138

ERRATUM

P-7584 *Differences between Building a Traditional DSS and an ODSS: Lessons from the Air Force's Enlisted Force Management System*, by Warren E. Walker, August 1989

This replaces the figure on page 33.

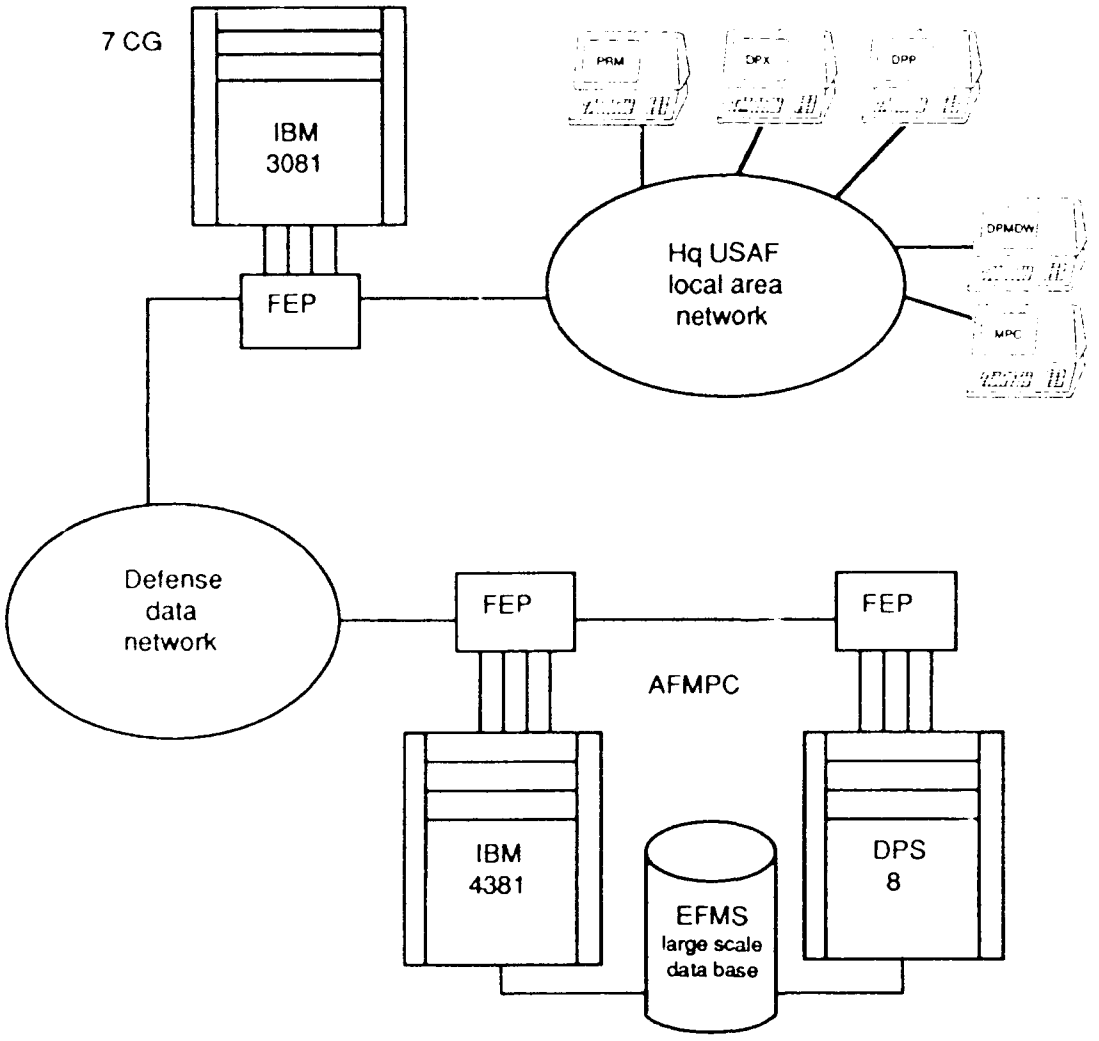


Fig. C -Force management system architecture

DIFFERENCES BETWEEN BUILDING A TRADITIONAL DSS AND AN ODSS: LESSONS FROM THE AIR FORCE'S ENLISTED FORCE MANAGEMENT SYSTEM[1]

Warren E. Walker
The RAND Corporation and Delft University of Technology

I. INTRODUCTION

Early descriptions of decision support systems (DSS's) were based on the paradigm of a single decisionmaker at a stand-alone terminal or microcomputer who had a specific decision (non-repetitive, semi-structured) to make. (Some of the early literature even recommended matching the user interface to the "cognitive style" of the decisionmaker). However, recent advances in computer technology, information systems, and telecommunications have facilitated a broadening of the scope of a DSS to include organizational units and, in some cases, entire organizations. In what follows, I will use the term "organizational decision support system" (ODSS) to refer to a DSS that is used by persons at several workstations in more than one organizational unit who make varied (interrelated but independent) decisions using a common set of tools. I will refer to the more traditional single user or single purpose system as a traditional decision support system (TDSS).

The basic building blocks of an ODSS are the same as those of a TDSS (see Fig.1):

- model base (and model management system)
- database (and database management system)
- user interface (a dialog system that manages the interaction between the user and the other two components)

[1]This paper was prepared for delivery at the 23rd Hawaii International Conference on Systems Sciences, Kailuha-Kona, Hawaii, January 2-5, 1990.

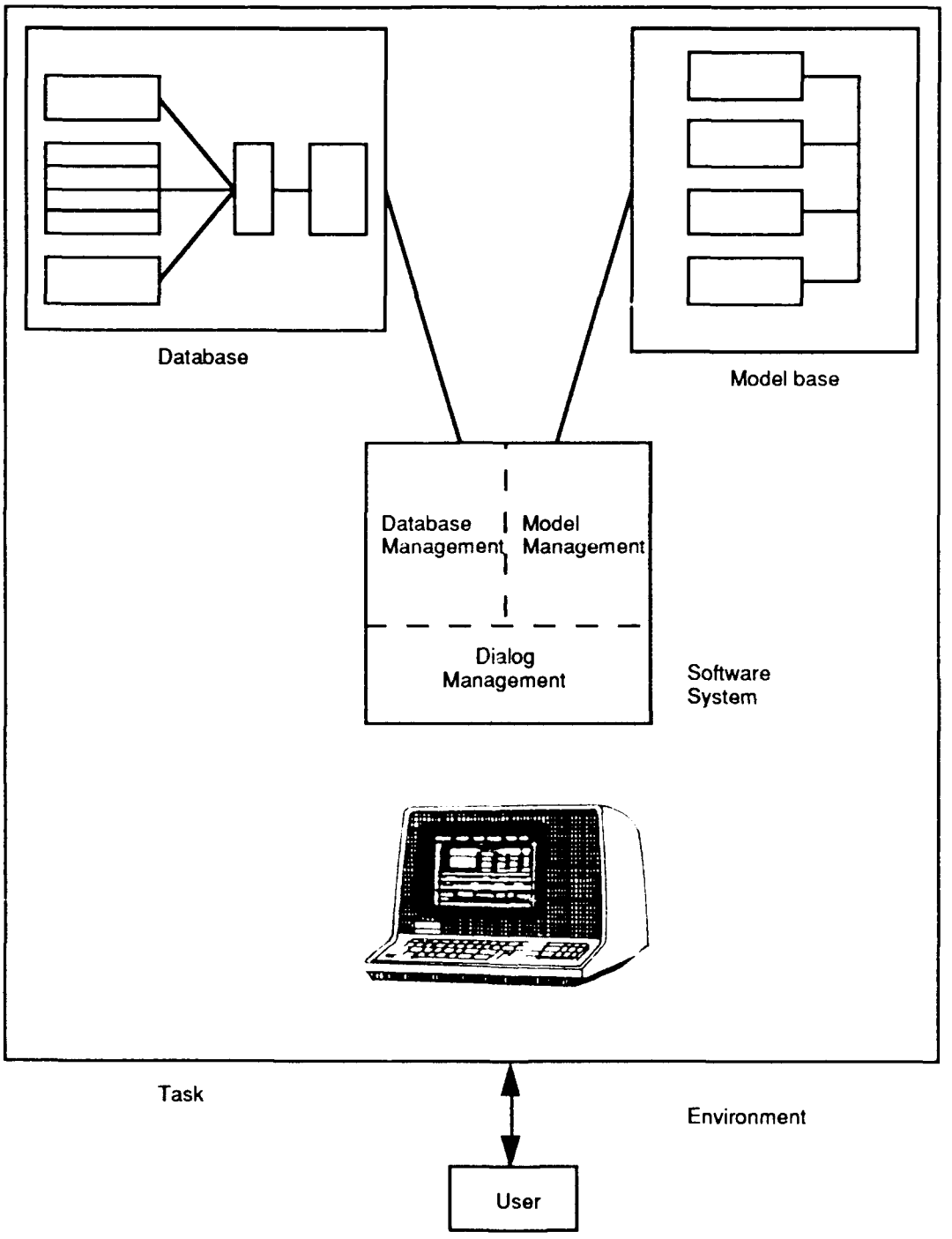


Fig. 1—Components of a DSS

But ODSS's are not simply big TDSS's. There are important differences. These differences lead to big differences in how ODSS's should be designed, developed, and maintained. In this paper I will discuss these differences, illustrating my points with examples from an ODSS that is currently being implemented by the U.S. Air Force. The system, called the Enlisted Force Management System (EFMS), is being used to help members of the Air Staff in the Pentagon make decisions related to their enlisted personnel. For an overview of the EFMS, see [Carter, et al., 1983].

Management of the enlisted force means making decisions about force structure, promotion policies, and the procurement, assignment, training, compensation, separation, and retirement of personnel. These functions (and their support activities) are spread among five major, somewhat independent, organizational units, led by four different two-star generals (three report to one three-star general; the other reports to a different three-star general). (See Fig. 2 for a chart showing the relevant organizations.) The five organizations reside in three geographically dispersed locations: three are in the Pentagon, one is at Bolling Air Force Base, about 20 miles away, and one is at Randolph Air Force Base in San Antonio, Texas. The EFMS was designed and developed jointly by a team of Air Force personnel and analysts from the RAND Corporation. The project (called the Enlisted Force Management Project, or EFMP) was begun in 1981. Implementation began in 1986, and is still continuing. To date the project has consumed over 120 man-years of effort.

I first discuss differences between the purposes of an ODSS and TDSS. The remainder of the paper highlights the major differences in how they are built and maintained.

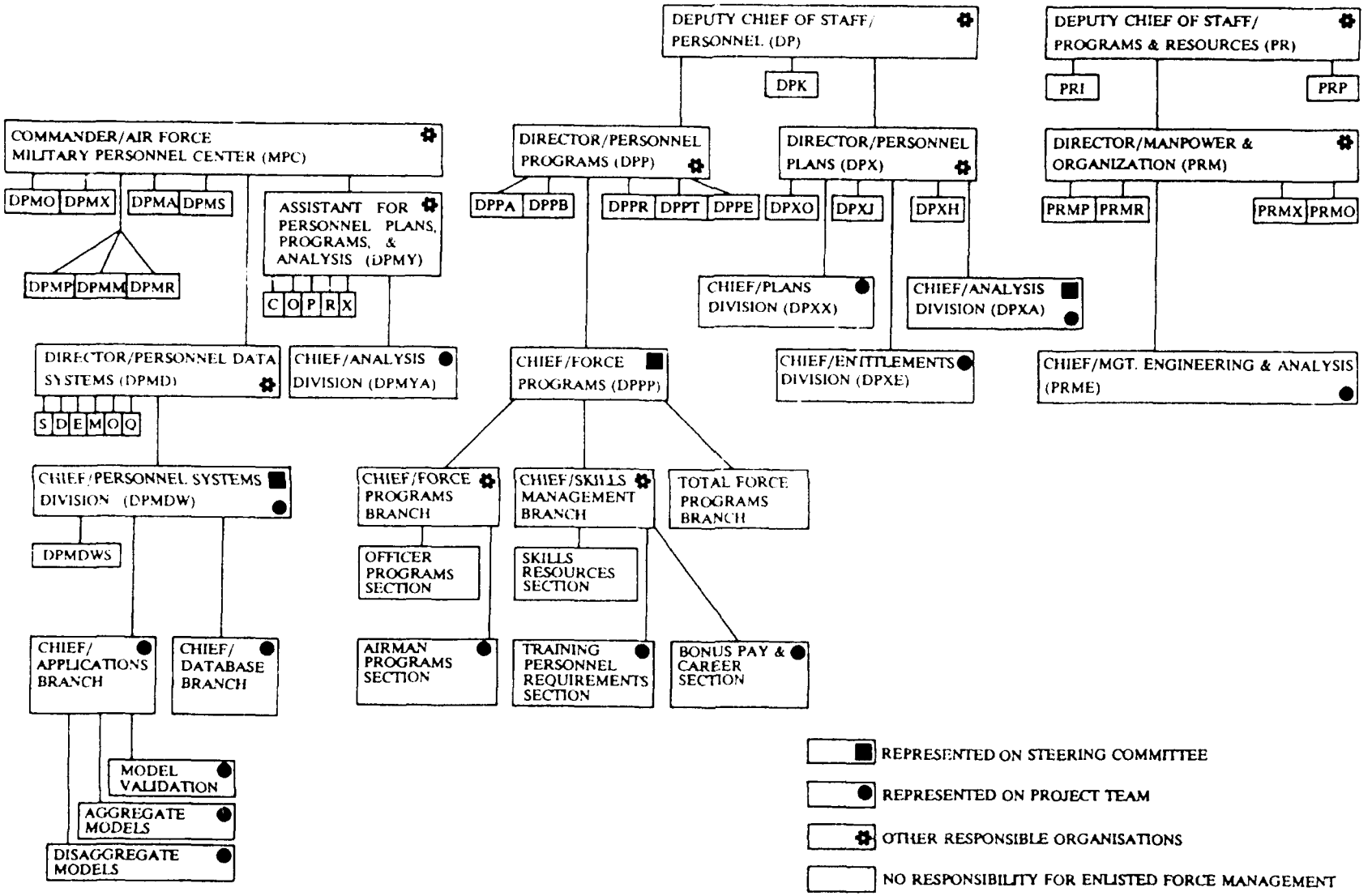


Fig. 2—Organizations responsible for enlisted force management in the USAF

II. PURPOSES

The primary purpose of a TDSS is to improve the performance of an individual decisionmaker—to improve the quality of his or her decisionmaking by improving its effectiveness and efficiency. This is certainly one of the purposes of an ODSS. However, its purposes are broader and more far-reaching. There are a multitude of organizational goals that a well-designed ODSS can benefit.

Most of these benefits result from the fact that an ODSS provides a means for communication and coordination among the various parts of an organization—among personnel at the same or different levels in the same organizational unit, and at the same or different levels across organizational boundaries. Since the entire organization shares a common database, as soon as a decision is made (e.g., the list of Air Force occupations that will be offered a bonus) the database is updated and the latest information can begin to be used immediately by others in the organization. This helps to integrate and unify an organization, and improves control and consistency. It also improves the efficiency and effectiveness of organizational decisionmaking (in addition to individual decisionmaking).

Bidgoli [1989, p.60] offers the example of a high-tech company. Before it implemented a cost-based DSS, different division managers would provide different prices for a finished product. The establishment of an on-line cost-based ODSS significantly improved the interpersonal communication and the intergroup coordination of these executives. His conclusion is that, by providing comprehensive information regarding the entire organization, "the overall control of the organization, including control over costs, inventory, and personnel, will be improved."

In the Air Force case, different organizations within the Air Staff used to use different models for projecting the future composition of the enlisted force. It was impossible to determine whether differences in the projections were due to different policies or different models. Now it is clear that differences in projections are due to differences in the policies.

III. PHASES OF DEVELOPMENT

The development of an ODSS requires a formal, structured approach, since we are talking about a large, complex, system programming effort. But this does not mean that it should be developed using a traditional system development life cycle (SDLC) approach (see, for example, [Lucas, 1985]), or that its design is frozen before work is begun. Most of the DSS literature is in agreement that such approaches are inappropriate for an environment in which users will learn more about their problem or environment as they use the system, and may identify new and unanticipated information needs. In these situations, an "iterative" [Sprague and Carlson, 1982, p.139], "adaptive" [Keen, 1980] approach is preferred.

The approach to building an ODSS that we recommend (and that we used on the EFMP) is a combination of the SDLC and iterative/adaptive approaches. It divides the process into four phases. The first two phases of the process are structured, and provide a framework for the development of the system. The third phase is iterative/adaptive, involving prototyping for the development of the system's modules. And the fourth phase uses both approaches. The phases are:

1. *Getting started.* This is the organizational phase, and is not iterated. It includes the following activities:
   - *Needs assessment.* What is wrong with the existing system and what would it take to solve it? What would be the objectives and goals of the ODSS? Does building an ODSS for these purposes make sense? (This is the "problem definition" stage of the SDLC approach.)
   - *Getting management support.* This means formulating functional recommendations for an ODSS, selling the idea to top management, and obtaining a commitment (and resources) to build the system. (This is the"feasibility study" stage of the SDLC approach.)
   - *Getting organized.* This involves setting up a steering committee and identifying the members of the project team (see Sec. IV).
   - *Getting a plan of action.* This involves laying out the plan for the development process. What steps must be carried out? What specific problems would the ODSS address (and what would it not address)?

In what order should the problems be addressed? How will team members communicate with each other? How will work get documented? How will decisions get made? (This is the "systems analysis" stage of the SDLC approach.)

2. *Developing the conceptual design.* This is the most important phase in developing an ODSS, and it is not iterated. This phase produces a blueprint for the system, which serves to guide subsequent decisions made by the system's builders. As Brooks [1975, p.42] says, "It is better to have a system omit certain anomalous features and improvements, but to reflect one set of design ideas, than to have one that contains many good but independent and uncoordinated ideas." (This is somewhat like the "systems design" stage of the SDLC approach, but the resulting design is less specific.) This phase is discussed in somewhat greater detail in Sec. V.

3. *Developing the system.* This phase is discussed in Sec. VI. It includes two types of activities:
   - *Designing the physical system.* This activity includes choosing the DSS Generator and other software, choosing the hardware, and designing the database.
   - *Developing the system's models and database.* Models are usually a more important factor in an ODSS than in a TDSS. But the approach to developing them is basically the same for both types of systems — prototyping.

4. *Implementing and maintaining the system.* This phase (which is a combination of the iterative and SDLC approaches, and which continues indefinitely) is discussed in Sec. VII. It includes:
   - Installing the physical system.
   - Programming and updating the system's modules. (We refer to the computer programs as modules and their mathematical specifications as models.)
   - Creating and updating the database.

— Documenting the modules and database.
— Training users.

Most of the first two phases should be carried out as if the system were being developed using a traditional System Development Life Cycle (SDLC) approach. This provides a framework and structure that will allow the flexible, adaptive, iterative, staged implementation of the system in the final two phases. Since there is an extensive literature on the SDLC, further details about the activities in these two phases will not be discussed. However, since the management of the development of an ODSS is so different from the management of the development of a TDSS (although it is consistent with an SDLC approach), we discuss this aspect in the following section; since the design of an ODSS is so different from the design of a traditional information system, we discuss aspects of the design in Sec. V. The final two sections deal with activities in the final two phases.

IV. MANAGEMENT

Building an ODSS is a much more significant undertaking than building a TDSS. In the latter case, the literature suggests that an ad-hoc approach can often be used, involving in some cases only the user (a "user-developed DSS" [Turban, 1988, p.133], a builder (programmer/analyst) and a user (see [Keen and Gambino, 1983, Fig. 7.5]), or a user with some help from an information center [Turban, 1988, p.121]. In contrast, an ODSS requires a much more structured approach—a team effort involving persons with a variety of skills at different levels of the organization and in different organizational units.

This difference can be compared directly to the difference that Brooks [1975] identified between writing a computer program and developing a programming system product. A program is "complete in itself, ready to be run by the author on the system on which it was developed." It might be able to be produced cheaply by one or two programmers in a short amount of time (say y man months). A programming system product is "a collection of interacting programs, coordinated in function and disciplined in format, so that the assemblage constitutes an entire facility for large tasks." Its individual programs must have been thoroughly tested and documented. In addition, its programs must be tested together in all expected combinations, and it must be designed to use only a prescribed budget of resources (e.g., memory space, computer time). According to Brooks, to make the original program (say, a TDSS) into a component of the system product requires 9y man months and, more important, requires the efforts of a team, not just one or two programmers.

What is needed is a structured approach involving two groups of people—a steering committee and a project team (discussed in this section)—plus a blueprint for the system, which we call its conceptual design (discussed in Sec. V). (Thierauf [1988, Fig. 7-2] suggests a similar approach. But he breaks the project team into two parts: the builders and the users.)

The *steering committee*, composed of top- and middle-level managers from all organizational units that will involved in building or using the ODSS, or that will be affected by it in some direct way, provides overall control and direction to the project. It makes policy decisions, assigns priorities, monitors progress, and allocates resources. It meets regularly (on the EFMP, the steering committee met quarterly). At these meetings, participants:

- see presentations by members of the project team (progress reports)
- get an understanding of the status of all work in progress
- decide on the schedule for future work
- resolve problems (e.g., allocate resources, assign priorities, obtain access to data)

The status of the members of the steering committee and its role in the project enables it to serve as a buffer between the project team and the organization's top management. It could easily promote the project with top management, disseminate information on the project's progress, and help smooth implementation of the system.

The *project team* is composed of staff members from all of the affected organizations (in roles as builders or users) plus, at its core, builders employed in a technically-oriented System Management Office (SMO). As in the case of a TDSS, development of a successful ODSS requires the active involvement and participation of the users from the very beginning. The system will be designed around them. They need to believe that it is "their system". Some organizational units (e.g., finance departments) also have their own analysts and programmers. The team should include them, because their knowledge of the users' problems will be helpful, and because they may be helpful in updating and maintaining the system. Some organizations may not have enough internal analytical support to design an ODSS. In this case, they might hire outside analysts to write the mathematical specifications for the system's model's. In the case of the EFMS, the Air Force asked RAND to play the lead analytical role. In particular, RAND's major responsibilities were to:

- develop a conceptual design for the EFMS
- develop the mathematical specification for all models in the system
- provide system programmers with advice on input formats and output reports
- provide advice on desirable hardware capabilities
- help the Air Force to implement the system and set up procedures for operating and maintaining it

But responsibility for the design, development, implementation, and maintenance of an ODSS should be assigned to the SMO, which is not one of the user organizations, but is an organization housing technical experts (software, hardware, statistics, etc.) whose sole *raison de etre* is to build and maintain the system and provide support to its users. This approach will help to reduce conflicts among organizational units.

The SMO should be set up at the very beginning of the project. Its head should be made the project leader. Its role on the project (and personnel numbers and expertise) will change over the course of the project, but it has things to do during all phases (design, development, implementation, and after implementation). During the design phase, it would be the responsibility of the SMO to:

- identify the specific needs of the various users of the system
- design the physical system
- select the DSS Generator
- select the system's hardware

During development and implementation, the SMO would:

- procure the hardware and software
- prepare the central computer facility and set up user workstations
- develop standards and procedures for programming, database management, and user interaction
- develop standards for documentation
- program the system's modules (including testing the modules)
- create the system's database
- document the modules and database
- develop training materials
- keep potential users informed of progress

As soon as one or more of the modules of the ODSS has been implemented, the attention of the SMO must begin to shift toward maintenance and updating. During development and implementation, the SMO, in consultation with the project team, would:

- train users (and answer questions as they arise)
- distribute hard-copy reports produced by the system
- update documentation of the modules and database as changes are made
- maintain and update the database
- maintain and modify the modules

The SMO is clearly more than a programming shop or MIS department. In addition to its technical responsibilities, the SMO (under the guidance of the project team) must play the role of a change agent, keeping the users involved throughout the development period and making sure that they understand what is happening and why. The work that this group does before implementation will determine to a large extent how successful the system will be. People in organizations are more or less resistant to change according to the way that change is introduced. To help improve chances for successful implementation, the SMO should consider such organizational and behavioral questions as:

- How will existing procedures be changed?
- Which jobs will be most affected and in what ways?
- How can the people affected be prepared for these changes?
- What sort of training will the affected people need?
- What is the best timetable for implementing the changes?

Some organizations already have an Information Center, which can serve as the SMO for an ODSS. Several companies (e.g., Northwest Industries, American Airlines, and Sun Oil Company) have formed DSS departments, which contain a group of people who have the necessary skills and capabilities to be DSS builders.

The talents required are many, which implies a large and diverse project team. Successful design, development, and implementation of an ODSS requires persons with an understanding of the problem area, plus expertise in such areas as mathematical modelling, statistics, database management, computer programming, organizational behavior, and human factors engineering. Leaving out any of these disciplines might lead to problems. And all of these perspectives on building the system should ideally be represented on the team from the beginning of the project, since it is hard to change direction once the project is under way.

Successful functioning of the project team requires continual interactions, good information flows, and close working relationships among team members. This is not so hard in the case of a TDSS, since the members of the team are in only one or two organizational units. But an ODSS project team includes persons from many organizational units who might even be geographically dispersed. The EFMS project team included persons in Washington, D.C., San Antonio, Texas, and Santa Monica, California. We used a wide variety of means for communication and coordination, including:

- trips by team members to other locations
- use of common computing facilities and common databases (via telecommunication)
- numbered memos distributed to all team members, accessible at all times in a book maintained by a designated person at each location
- exchange of information by telephone, overnight mail service, facsimile, and electronic mail
- use of an action plan, defining tasks, schedules, and responsibilities

The most efficient and effective way of organizing the project team is likely to be the "surgical team" approach recommended by Brooks [1975] for organizing a system programming team. In this approach, a single person (the "surgeon") is made responsible for a module of the system. He personally defines the functional and performance specifications, designs the program, codes it, tests it, and writes its documentation. Other members of the team with specific expertise (e.g., database management, statistics, programming) provide support to this effort. The success of this approach depends upon starting out with conceptual design of the entire system that has "conceptual integrity"—i.e., it is clear what the pieces are and how they relate to each other. If this approach is used, it is then possible to have separate teams working independently to develop different modules, and still have them integrated into a coherent system.

V. CONCEPTUAL DESIGN

The conceptual design of an ODSS tells what should happen; implementation tells how it is made to happen. According to Brooks [1975, p.44]:

> For a given level of function, . . . that system is best in which one can specify things with the most simplicity and straightforwardness. . . . Simplicity and straightforwardness proceed from conceptual integrity. Every part must reflect the same philosophies and the same balancing of desiderata . . . .Ease of use . . . dictates unity of design. . . . The separation of architectural effort [conceptual design] from implementation is a very powerful way of getting conceptual integrity on very large projects.

Once it has been decided that an ODSS will be built, but before any work is done on specifying hardware, software, or models, it is necessary to produce a conceptual design. The design must include at least the following elements:

- Design principles, which will guide all decisions for the remainder of the project
- Functions to be supported
- Models to provide the support, how the models would work (including inputs and outputs), and the relationship among the models (e.g., a flowchart showing interconnections)
- Data requirements (generic data; no file names or database layouts)
- Hardware and software considerations (hardware configuration and software capabilities, not specific equipment and languages)
- Approach to implementation (structure of SMO, priorities, prototyping strategy, documentation rules, responsibilities of participating organizational units)

DESIGN PRINCIPLES

Five principles guided the design of the EFMS. These principles are widely stated in the DSS literature, but are easier to say than to do. By stating them before any development work was begun, they guided the entire system development process. The principles were:

1. Improve the effectiveness and efficiency of enlisted force management and decisionmaking
2. Place the user in control
3. Make the system fast, inexpensive, and easy to build
4. Make the system flexible, adaptable, and easy to maintain
5. Coordinate and integrate the decisionmaking environment

The first four principles apply equally well to TDSS's. Principle 5 is one of the most important reasons for building an ODSS instead of a separate TDSS for each function.

These principles, although traditional and obvious, had important implications for the design of the three basic components of the EFMS: the database, model base, and user interface.

Model Base

The desire for flexibility, adaptability, and easy maintainability suggested the use of an interlinked system of many small models (or modules), each designed for one specific purpose, instead of a few complicated, comprehensive, large models. A module can be used by itself to study the impacts of a proposed decision on a specific portion of the organization, or interactively with other modules to study the wider impacts.

The modular approach to modeling is attractive for a variety of reasons. In addition to mitigating the problems inherent in building a single large model, it makes it easier for users to understand (and accept) the models in the system. The modular approach also makes it relatively easy to adapt to a wide variety of circumstances, availability of data, and types of analyses without having to incur large amounts of time, skill, and confusion in reprogramming.

Modules also make it easier to use the "surgical team" approach to writing the DSS computer programs. As Miller and Katz [1986] explain: "The various modelers need communicate about the inputs and outputs of the submodels for which they are responsible, but they do not have to understand each other's submodels in detail."

Database

The desire for coordination and integration led to the specification of a common, consistent, easily accessed, centralized database for the system. The database would provide input to the modules, would retain output from the modules for management reports, and would be available for direct inquiry by users. Information generated by one module would be automatically (and instantaneously) available to other modules. Data both internal and external to the organization would be included. (For example, the EFMS database includes information on the inventory of airmen and data on the U.S. economy.)

The system need not have a single, unified, integrated database. In the EFMS, each module currently has its own database. But database administration should be centralized (assigned to the SMO), responsibility for updating and maintaining each item of information should be assigned, and each piece of information should be stored in only one place. Also, since the system is being used by different organizational units, there must be privacy and security provisions built in. Some users would be unable to access certain information and those permitted to change data in the central database would be specifically linked to those items of data for which they were responsible.

User Interface

The primary implication of the design principles for the user interface is that the system have a common interface for all of its elements; that is, that dialogues be managed in a uniform fashion regardless of the particular module being run. Of course, each module would have different specific input and output screens. But each would enable the user to do the same types of things in the same ways. For example, if the user wanted to run a module, modify a parameter value, or name an input data file, he would use the same procedure for all modules. Also, since the users of the EFMS were not programmers, we decided that the interface should be menu driven, easy to learn, and easy to use. The user (without the help of a programmer) would be able to

- request information from the database
- make temporary or permanent changes to data in the database
- specify parameters and input data for a module
- run a module
- tailor output reports (scope, aggregation, time periods)

FUNCTIONS TO BE SUPPORTED

This is another major difference between the focus of TDSS and ODSS development. The focus in the development of a TDSS is usually on the individual decisionmaker. Screens are often designed by him or to his specification, and interface protocols (e.g., menus, logic, etc.) are designed to match his decisionmaking style. This is the view of a TDSS that Wagner [1981] refers to as Executive Mind Support. According to him, "Executive Mind Support is the intimate coupling of a system (a DSS) with the mind of an executive, with close rapport and two-way communication, for the executive's own purposes and on his own terms. It is a special relationship by which the system actually supports and extends the manager's own thinking processes."

The focus in the development of an ODSS is on the functions to be performed. The ODSS is part of a unified, organizational approach to problem solving. It must, therefore, be designed with a consistency and unity that is inconsistent with a focus on individual differences. Also, as Huber [1983, p.575] suggests, each function in an ODSS is likely to have different users over time, as job incumbents move through a position. "Thus a flexible rather than an idiosyncratically constrained design seems called for." In the words of Carlson [1979, p.19], "if a DSS is to support varying styles, skills, and knowledge, it should not attempt to enforce or to capture a particular pattern."

Thus, on the EFMP, the conceptual design document was devoted almost exclusively to a functional description of the system—the constituent modules and their interconnections. The user interfaces were designed in conjunction with the incumbents in the user positions, but the idea was to make sure that all of the information needed by a person in that position would be available in an understandable and usable format. Screens were designed assuming "average" user behavior, so the modules would be useful to practically any user (of course the screens can be changed if a new user doesn't like something). Because of officer rotation, several of the modules have already had several "owners".

By focusing on functions, the builder of on ODSS can avoid two other pitfalls that can lead to the failure of the system. First, the pitfall of designing the system to reflect current organizational responsibilities and interactions. Some systems are designed too closely around an existing organizational configuration. Organizations change frequently (e.g., departments move around on the organization chart), but the functions that have to be carried out within the organization are more stable. So, if

modules are designed for functions not organizational units, use of the modules can be transferred to new organizational units when the functions are transferred. At the beginning of the EFMP, the Directorate of Manpower and Organization (see Fig. 2) reported to the same three-star general as all of the other organizations using the EFMS. In the middle of the project, the "manpower" functions were split off from the "personnel" functions, and assigned to a different three-star general. This had no effect on the design or implementation of the EFMS.

Second, focusing on functions can avoid the pitfall of using the implementation of a system for rationalizing decisionmaking as an excuse for trying to rationalize the structure of the organization. In studying the existing system in Phase 1 of the ODSS development process, it will probably become clear that improvements in the efficiency and effectiveness of decisionmaking can be made by changing the structure of the organization. It is tempting to write a report that recommends that this be done in addition to the construction of an ODSS, and that the design of the ODSS be made dependent upon the new structure. Unless your tasking for Phase 1 explicitly includes a requirement for producing recommendations on organizational design, you should steer clear of this. Focusing on functions allows you to do so. The first RAND project leader on the EFMP (I was the second) recommended a consolidation of the three directorates responsible for manpower and personnel decisionmaking as part of the EFMS design. As a result, the general in charge of one of the directorates tried to cancel the entire project. (You can submit a separate report making some suggestions for organizational changes. But this should not be tied to the ODSS development effort.)

MODELS

The conceptual design of an ODSS should include a listing of the modules the system will eventually contain, a short description of the function of each one, their major inputs and outputs, and a flowchart showing their interrelationships. Just as it is useful in forming the project team to forecast the needs to the end of the project and include all necessary skills from the beginning, it is important to try and specify the full functional scope of the system at the conceptual design stage. This will make it easier to construct a coherent, integrated, unified system. Without this framework, it is likely that the system will end up a fragmented patchwork-quilt, held together by baling wire. With this framework, the system can be developed in an iterative,

adaptive way. Whenever a module is ready to be added to the system, it can be plugged in, and the connecting hooks will be in place to greet it.

Figure 3 is the summary flowchart for the EFMS that appeared in the conceptual design document [Carter, et al., 1983]. It indicates the four major sets of modules in the system, their major inputs and outputs, and their interrelationships. The conceptual design also included more detailed flowcharts for each of the sets of modules, which showed each specific module, its inputs, its policy levers (i.e., decision variables), its outputs, and where the outputs were to be sent (e.g., reports to Congress or the Department of Defense).

DATA

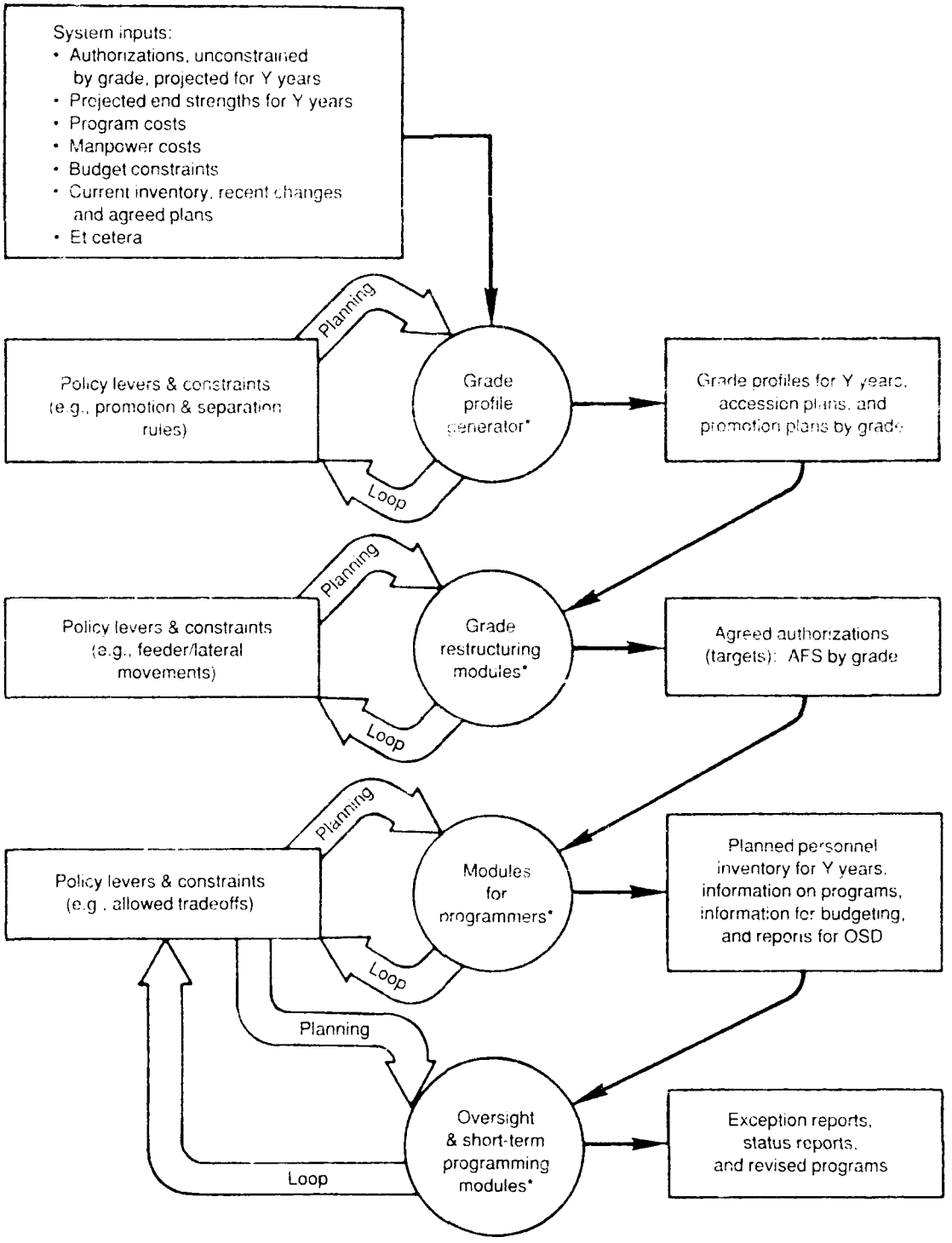
An ODSS has much more demanding needs for data than does a TDSS, and more attention has to be given to this aspect of the system. In general, there are four different types of data that are used during the course of building an ODSS. These are data:

1. to understand or define the problem situation being addressed
2. to estimate the models
3. to validate the models
4. to run the models (input data)

We call the data for the first two uses "analysis files", since they are primarily used by analysts engaged in defining and building models. The other two types of data are "operational files", which are created and used by the SMO.

A great deal of thought must be given to data issues at the conceptual design stage, and a great deal of time must be allocated to creating these databases during the development of the system. Projects that make extensive use of large data files usually underestimate the amount of effort required to create useful databases. We estimate that between 25 percent and 40 percent of the effort on the EFMP was devoted to collecting, cleaning, and analyzing data.

The conceptual design document should try to estimate the amount of data that will ultimately be required within the system, where the data would come from, and a strategy by which data files can be added to the system as modules are added without affecting the integrity or the consistency of the database.



*including an inventory projection module in which loss rates are calculated as functions of historical patterns, Air Force policies, and external conditions

Fig. 3—Summary flowchart of the EFMS

A principle that we included in the conceptual design for the EFMS for making the system easy to update and maintain was that the data required by the modules should be as easy to obtain as possible. We said that the input data should not require extensive preparation or previous analysis and should be routinely collected by the Air Force or some stable external source (such as the Census Bureau or Department of Labor).

HARDWARE AND SOFTWARE

Although the functional specification of an ODSS can be accomplished without considering the physical environment within which it will be implemented, it is still important to include such considerations in the conceptual design document. At this point, only the general outlines of the hardware and software required to support the system are to be specified. This information serves two purposes:

1. The organization's management needs such information in order to estimate the system's development cost
2. The system's users need such information to understand how the system will work and how they will fit into it.

Since the system will be supporting users throughout the organization, it is likely that one of two types of distributed data processing approaches would be used: (1) a mainframe linked with PCs or user workstations through a local area network (LAN), or (2) user workstations linked with a central fileserver through an LAN. The general physical configuration specified for the EFMS is shown in Fig. 4. End users, in geographically dispersed sites, would utilize microcomputer workstations. Through a high-level, English-like command language, he or she would interact with both an integrated database and an interlinked system of modules, both of which would reside on a mainframe computer. The software would combine a wide range of capabilities in a single package (see Fig. 5). Both Figs. 4 and 5, which are part of the conceptual design, show generic functions and capabilities. The actual selection of the hardware and software comes during the development phase (see Sec. VI).

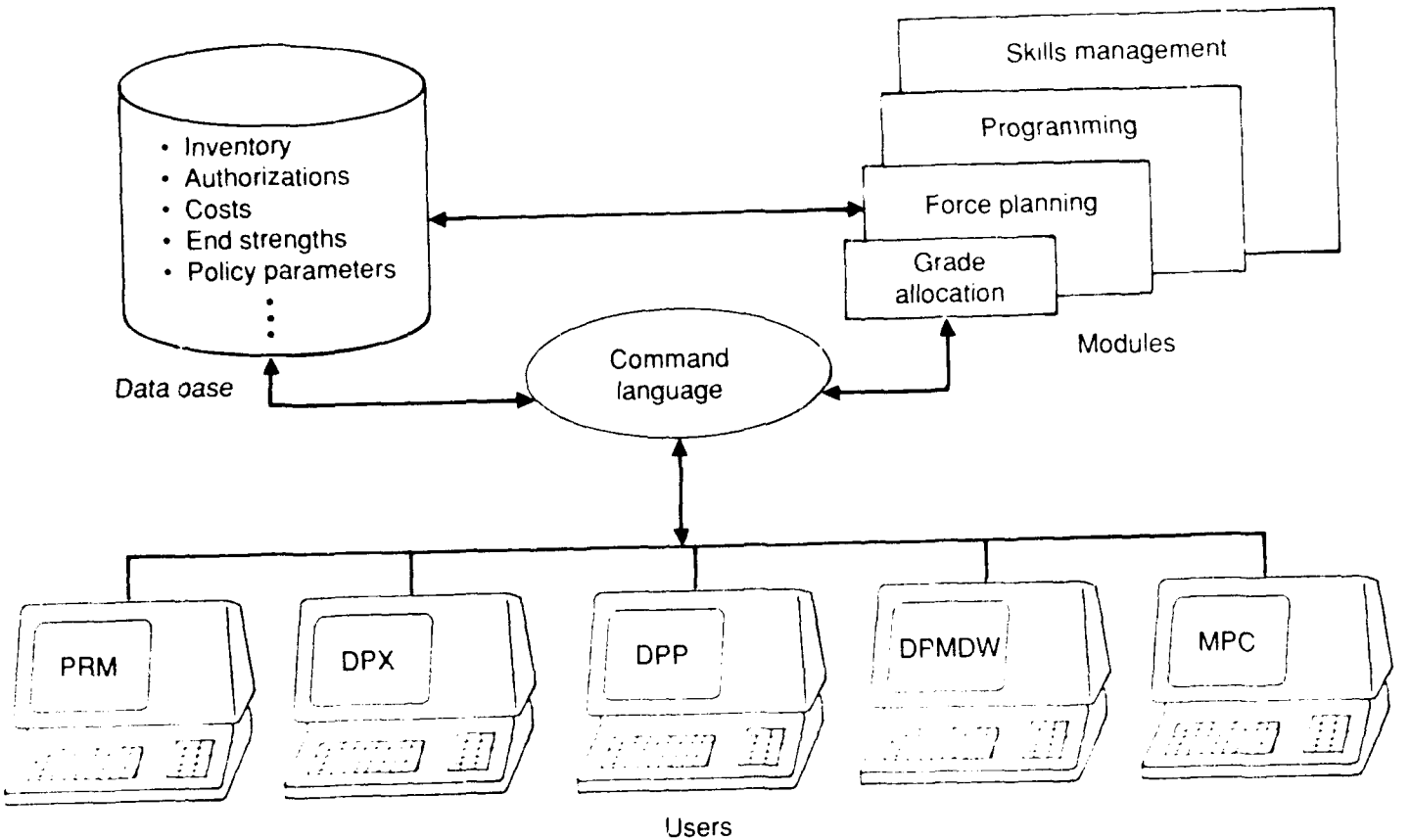


Fig. 4—Major elements of the EFMS

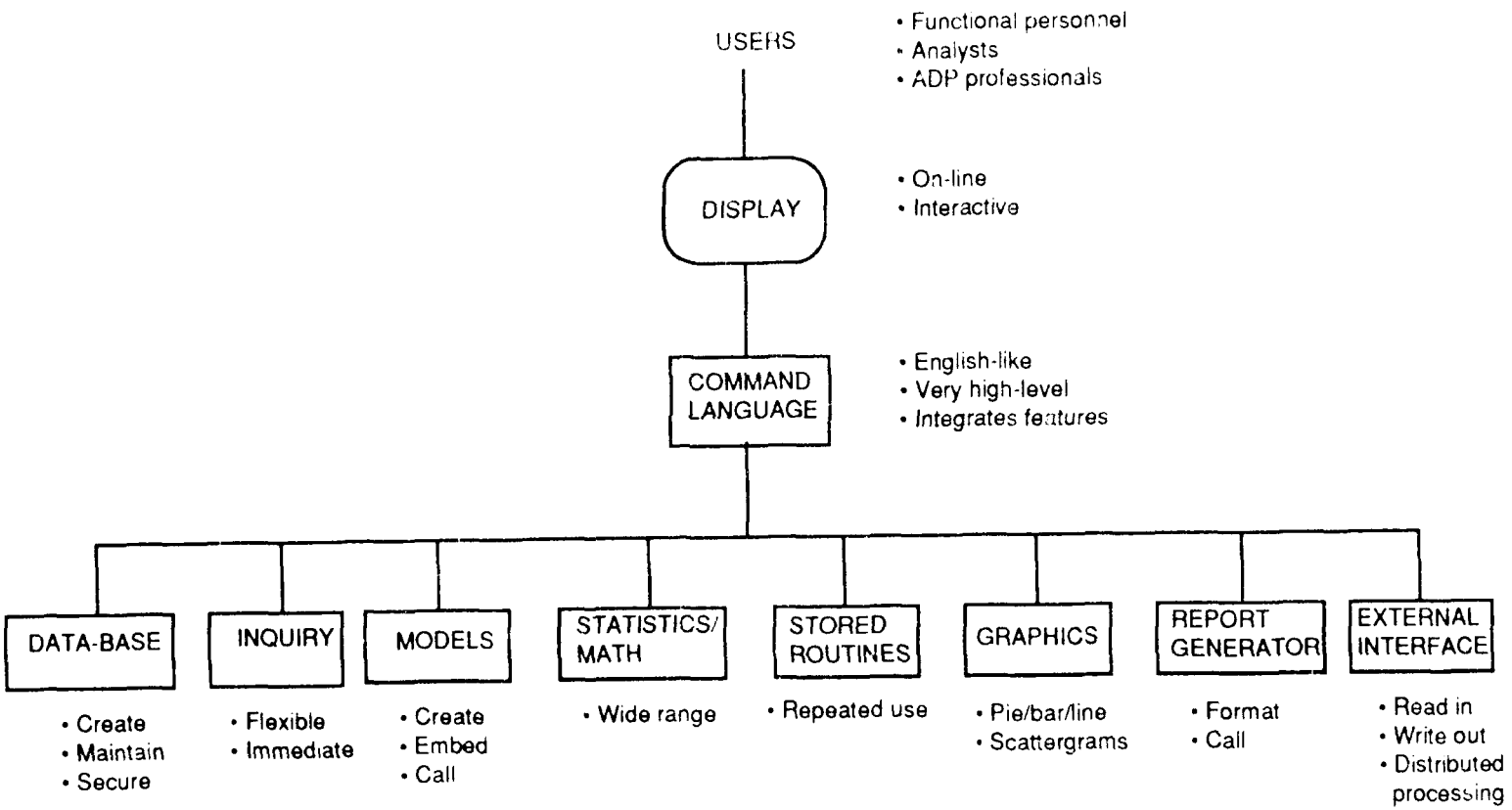


Fig. 5—Elements of the EFMS DSS generator

Developing, Implementing, and Maintaining the System

Up until these last two phases, the building of an ODSS has been carried out according to the traditional System Development Life Cycle approach for building information systems. But the development, implementation, and updating of an ODSS apply the middle-out [Hurst, Ness, Gambino, and Johnson, 1983], iterative [Sprague and Carlson, 1982], adaptive [Keen, 1980] principles found in the DSS literature. Middle-out development is based on building and using early prototypes of the modules to get quick feedback on their form and substance. Iterative design involves continual improvements to the system until the users' requirements are satisfied. Adaptive design advocates a process of continually modifying the DSS to meet changing needs and conditions. We called our approach to building the EFMS "staged implementation".

In staged implementation, some modules are developed in parallel with others, and some are developed sequentially, in priority order. Use of a module can begin whenever it has reached the point that a user feels comfortable trying it. In addition to the implementation of modules one at a time, development of each module is an iterative process involving several members of the project team (at least an analyst, a programmer, and the user) that includes some or all of the following:

- conceptual design (with the user intimately involved)
- mathematical specification (which includes mathematical modeling, estimation of the parameters of the model, and validating the model using historical and/or hypothetical data)
- programming a stand-alone prototype of the module
- testing and using the prototype for some or all of its intended functions (the user is the main participant in this step)
- evaluating the test
- revising and improving the mathematical specification (which includes the possibility of adding features to the model)
- reprogramming the module for inclusion in the system
- preparing the database that is needed for updating and reestimating the model
- integrating the module into the system (which includes adding its input data files to the system's database)

All of these steps would not necessarily be carried out for each module, and the development of each module would not necessarily involve carrying out the steps sequentially. There would be a lot of iteration and feedback among the steps. For example, testing of the prototype might reveal problems that would return development of the module to any of the previous three steps (even rethinking the conceptual design).

For smooth and efficient implementation, it is best to have the prototype built and tested by the same group that developed the mathematical specification (not by the system programmers in the SMO). Then, after the prototype is found to work well, the system programmers can reprogram it. This enables errors in model specification to be separated from errors in programming.

What I have described above is the "throwaway" approach to prototyping [Sprague and Carlson, 1982], in which, once the prototype is working as the designers and users want, it is reprogrammed for inclusion in the system. There is also an "evolutionary" approach, in which the prototype is modified to integrate it into the system. We used both approaches on the EFMP. It is tempting to use the evolutionary approach ("if it works, don't fix it"). But I recommend the throwaway approach, since the prototype is usually not written for efficiency or maintainability. (It may contain patchwork fixes, it may be hard to document, its data structures may be inefficient, etc.)

The prototypes are likely to include some, but not all, of the features of the final versions of the modules. The inputs, outputs, and user interactions of the prototypes might be different from those planned for the final system. But there are several good reasons for using them in their early versions:

- Support for some areas of decisionmaking can be obtained early in the system development process.
- Ideas can be tried out without incurring large costs
- Problems with the modules can be identified and corrected early in the process.
- Users can gradually become familiar with the concepts, procedures, and modules of the system, and, because they are participants, they will be more likely to get exactly what they need.
- The System Management Office can gradually build up its organization and procedures.

As Jenkins [1983] has found (based on examining over 120 prototype systems), prototyping usually results in greatly reduced systems development time, lower overall development costs, increased user satisfaction, and effective utilization of scarce resources (inefficiencies are tolerated in machine utilization, not in people utilization). He concludes that the success of prototyping is based on a simple proposition: "people can tell you what they don't like about an existing system easier than they can tell you what they think they would like in an imaginary system."

VI. DEVELOPMENT

I have described a great deal of the recommended approach to developing an ODSS in the preceding sections. In this section I reiterate some of the points made above, plus I describe an approach for choosing the system's software that is one of the most critical steps in the development phase.

Successful development of an ODSS requires a great deal of cooperation, communication, and intense effort among all members of the project team, plus strong, clear guidance from the steering committee. Each of the actors has an important role to play, and none can be omitted. On the EFMP, the major actors were:

- The SMO (which included analysts, system programmers, database builders, and hardware specialists)
- Potential users of the system, in various manpower and personnel organizational units
- Analysts in various manpower and personnel organizational units
- The RAND Corporation (which supplied analysts for helping to specify the conceptual design, for specifying the models, and for building prototypes)
- The steering committee (which was directed by the head of the SMO and RAND's project leader, and included other managers from the SMO and various user organizations)

After the conceptual design document was written and agreed to by all parties, a range of activities were begun. The steering committee specified development priorities, RAND analysts were assigned to the various models, users were interviewed about their needs, requirements for analysis data were formulated, analysis data files were constructed, models were estimated and validated, prototypes were built and tested, and the system's hardware and software were chosen. In this paper I discuss only issues related to the creation of analysis data files and the selection of the system's software, since most of the other issues apply equally well to TDSS's.

CREATING ANALYSIS DATA FILES

ODSS data files (source data files, analysis files, and operational files) are much larger than those in a TDSS. Source data are used to create both of the other two types of files. But the source data usually consist almost entirely of secondary data—i.e., data collected by others for purposes different from those of the ODSS. An important (and time-consuming) task is to understand these data, clean them, and use them to define other variables that are more useful for the purposes of the ODSS. About one-third of RAND's effort on the EFMP was devoted to creating clean and useful analysis files.

The process of using data to estimate models can be broken into two phases: the audit phase (which embraces all steps in cleaning the data and increasing the modeler's understanding of the data) and the analysis phase (which includes data analysis and model fitting). Relles [1986] says that projects involving the analysis of large data sets usually allocate about 60 percent of their resources to the analysis phase, and only 40 percent to the audit phase. He suggests that it would be more efficient and effective to allocate about 65 percent of the project's data-related resources to the audit phase, including more time from the project leader and a senior programmer. This is because it is hard to catch errors in large data files, but errors can be costly if not caught early, and they may undermine the quality of the models if they remain undetected. He suggests a systematic approach for data cleaning and file creation, many aspects of which we applied on the EFMP.

SELECTING AN ODSS GENERATOR

An important principle for building a DSS (either a TDSS or an ODSS) is to choose the software before choosing the hardware, if at all possible. Tailoring the system to the problem situation and the needs of the user requires providing a set of specific capabilities. There are likely to be few software products available that provide all of these capabilities. By adding hardware constraints, the number of possibilities is reduced even further, leading to the use of a product that may seriously compromise the performance of the system.

Sprague and Carlson [1982] identify three types of software/hardware that are included in the label "DSS".

- *Specific DSS.* This is the combination of hardware and software that helps a specific decisionmaker or group of decisionmakers deal with

specific problems. It is the product that we are concerned with in this paper—the end result of the system development effort that we have been discussing.

- *DSS Generator.* This is a software package that provides a set of capabilities to build a specific DSS quickly and easily. It generally provides most of the software capabilities needed by the specific DSS. The choice of the DSS Generator is a very important decision. We discuss an approach for making this decision below.
- *DSS Tools.* These are individual hardware or software elements that can be used to develop either a Specific DSS or a DSS Generator. In general, development of a Specific DSS from a DSS Generator is faster and more economical than using DSS Tools (just as writing a computer program is usually faster using FORTRAN than machine language).

A Specific ODSS will normally be built by system programmers using a DSS Generator and (perhaps) other DSS Tools. Most users will never learn about most of the capabilities of the Generator or the Tools. This situation is distinctly different from the relationship between DSS Generator and user that is often recommended for a TDSS. In this case, the user is often expected to build his own Specific DSS using the modeling language provided by the DSS Generator. (For example, Reimann and Waren [1985] say that one of the purposes of a DSS Generator is to "enable nonprogrammers to develop customized DSSs for specific applications.")

In the case of an ODSS Generator, some capabilities will be helpful to end users, some only to the systems programmers, and some to both. For example, it should provide the benefits of a nonprocedural language to users to make it easy for them to operate the models, generate reports, and make inquiries against databases. It should also insure that systems professionals have all of the facilities within the package to write complex applications programs without having to resort to other packages (e.g., for graphics, statistics, or modeling) to a large degree. This demands a language that combines simplicity for one category of user with a powerful and varied syntax for another.

The process of choosing an ODSS Generator is similar in some respects to that recommended for choosing a TDSS Generator. But there some major differences. For a TDSS Generator, it is recommended that end users be extensively involved in the process from beginning to end, that the capabilities be directed toward the needs

of the end user, and that the end user control the selection process (see [Meador and Mezger, 1984] and [Reimann and Waren, 1985]). For an ODSS Generator, I recommend little end user involvement, capabilities directed toward the needs of the application, the SMO, and the system programmers, and SMO control of the selection process.

Since it is such an important decision, and since there are potentially so many requirements and so many alternative packages to be evaluated, a structured approach should be used to choose the ODSS. The process that we used on the EFMP is similar to that suggested above for the first two phases of building an ODSS. It is described in detail by Walker, Barnhardt, and Walker [1986]. The basic idea is to carefully match the specific features and capabilities of the generators under consideration with the characteristics and requirements of the applications to be supported. The approach involves six steps:

1. Identify the overall objectives for the generator (what it should accomplish and why).
2. Infer the general capabilities that the generator must have to respond to the objectives. (See [Turban, 1988, p.203] for a fairly extensive list.) The general capabilities are likely to be very similar for most ODSS situations. (E.g., provide a common database manager and allow customized menus.)
3. Infer a set of specific capabilities that will satisfy the general capabilities. The specific capabilities will generally differ for different applications. (E.g., allow the use of data names that provide consistency with the organization's naming conventions.)
4. Identify specific software products that have appear to have some or all of the specific capabilities.
5. Perform an initial screening of the products that are obviously over- or under-qualified (if over-qualified, you will be paying too much to satisfy your needs). Screening can usually be done by using reference services, reading product documentation and/or having vendors demonstrate their products and answer questions about their capabilities.
6. Perform a detailed analysis of each of the remaining products. This phase should be a systematic examination of product capabilities against requirements. It might also include benchmark runs, coding of test

problems, an analysis of the reliability of the vendor, and the likely life of the product and its support system.

On the EFMP, we identified ten general capabilities that the ODSS Generator for the EFMS should have:

1. Data management (the ability to build, maintain, and manipulate complex data structures, to provide access to information in a flexible and responsive manner, and to facilitate use and sharing of data)
2. External interfaces (ways to transfer data into and out of the system, and the provision of hooks to other programming languages (e.g. SAS, FORTRAN))
3. Data analysis (facilities for the statistical analysis of data)
4. Inquiry (an interactive database inquiry facility that would allow users to selectively view the data they need for a given task)
5. Report generation (default formats and customization)
6. Graphics
7. Command language
8. Multi-user support
9. System management facilities
10. Support for distributed data processing

Most of these capabilities are displayed graphically in Fig. 5. Note that all but the last three of the general capabilities are ones that are likely to be specified for many TDSS's. The last three apply primarily to ODSS's.

After specifying these general capabilities, we defined specific required capabilities within each category. For example, there were four specific required capabilities within the Multi-user Support category, including "Provide safeguards for the security and protection of data at the record level or below."

Then we began the search for and selection of a DSS Generator, which involved the following steps:

- Reading technical publications and systems documentation
- Interviewing system users and talking to vendors
- Screening (12 of 20 products were screened out)

- Detailed analysis of the remaining eight products
  - — Rating each product (yes/no) on each specific capability
  - — Giving a summary rating for each product on each of the ten general capabilities
  - — Comparing the summary ratings of all eight products across all ten categories (Table 1 shows this summary scorecard)
  - — Performing a benchmark test on a sample application

Table 1
SUMMARY EVALUATION FOR DSS GENERATORS

	Express	DSS A	DSS B	DSS C	DSS E	DSS F	DSS G	DSS H
Data management	Yes	No	No	No	No	No	No	No
External interfaces	Yes	Yes	Yes	No	No	No	No	Yes
Data analysis	Yes	Yes	Yes	No	Yes	No	No	No
Inquiry	Yes	No	Yes	No	No	No	No	No
Report generation	Yes	No	No	No	No	No	No	No
Graphics	Yes	Yes	No	Yes	Yes	Yes	Yes	No
Command language	Yes	No	No	No	No	No	No	No
Multi-user support	Yes	No	No	No	No	No	No	No
System management	Yes	No	No	No	No	No	No	No
Distributed data processing	Yes	No	No	No	No	No	No	No
Meets all criteria	Yes	No	No	No	No	No	No	No

Once the search was begun, it took approximately two years until the necessary approvals were received to purchase the selected product. Three important factors, other than our desire to do a careful and thorough search and evaluation, contributed to the time and effort required to acquire the DSS Generator: federal procurement directives, the large number of products, and internal resistance. The last is a factor that is likely to be encountered by most builders of ODSS, and is not generally addressed in the DSS literature.

There was considerable organizational reluctance to accepting a DSS Generator as a means of developing and operating the EFMS. Part of this reluctance stemmed from the fact that DSS Generators were a new concept to many, and a certain amount of education was required before gaining acceptance of the idea. The selection of the Generator was driving the supporting hardware options, and the

Generator chosen required equipment that was incompatible with the systems then in use. There was also concern about assuming the additional burden of operating and maintaining these new computer systems. Also, historically, the emphasis had been on developing and maintaining computer systems that emphasized transaction processing. The unique needs of the Air Staff—which emphasized flexibility, user control, quantitative and analytical capabilities, responsiveness, and the use of

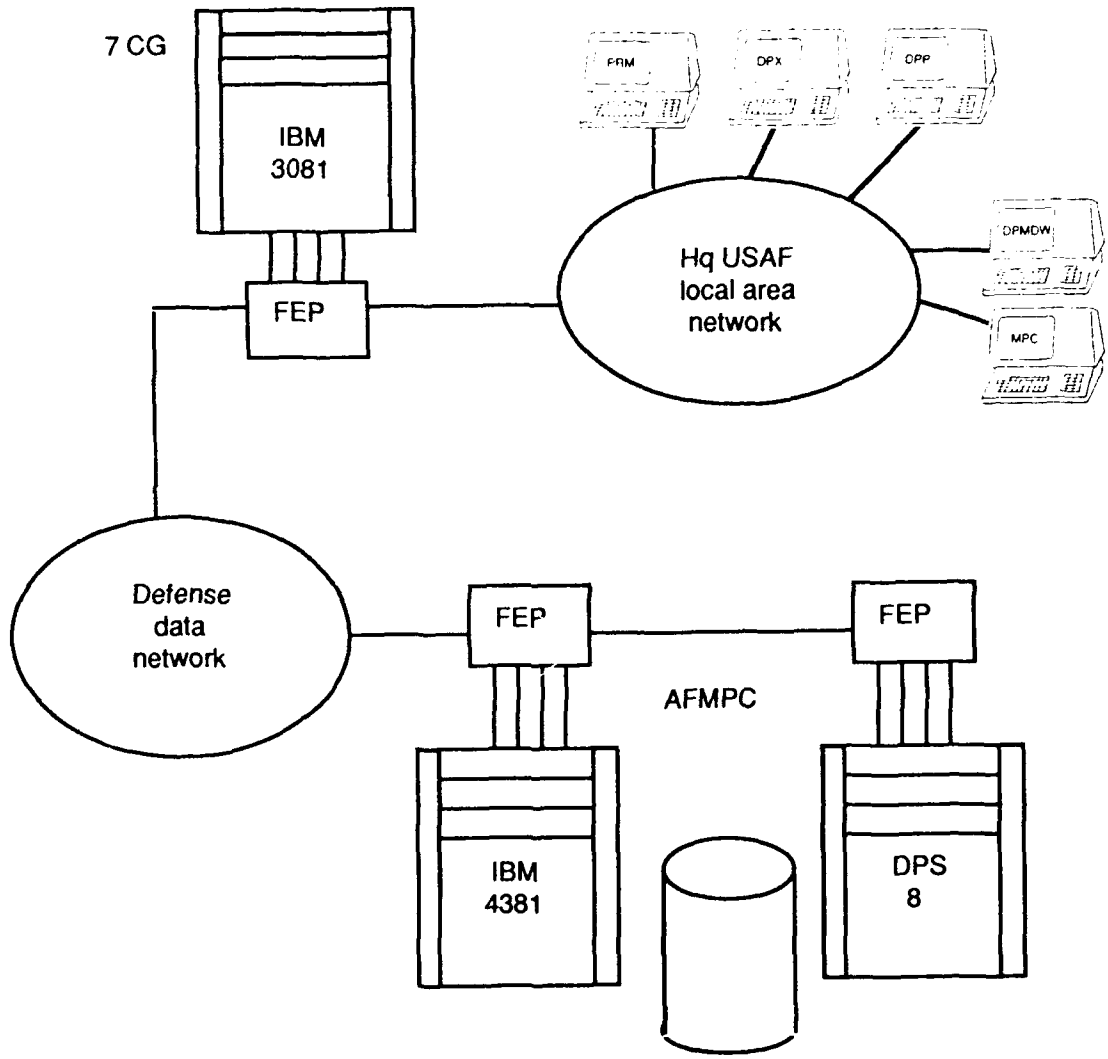


Fig. 6—Force management system architecture

summary data—were not immediately apparent. In the end, the compelling nature of the arguments won most people over, and many Air Force personnel worked diligently to acquire and implement the Generator.

Once the software was selected, the system's hardware configuration was able to be specified in detail (see Fig. 6 for an overview of the hardware configuration), and the process of procuring both software and hardware was begun.

VII. IMPLEMENTATION AND MAINTENANCE

The recommended approach for Phases 1 and 2 of the process of building an ODSS basically followed the System Development Life Cycle (SDLC) approach. Phase 3 departed from this approach, following instead the approach recommended for building a TDSS. The recommended approach for Phase 4 is a combination of the two approaches.

Many of the characteristics of an ODSS are similar to those of a traditional information system—e.g., large data files, high activity levels, and many users. Thus, some variation of the SDLC approach is necessary and appropriate. In particular, good SDLC practices for procuring and installing computer hardware, creating standards and procedures for programming, creating databases, documenting programs and databases, and training users are also applicable to an ODSS.

However, unlike a traditional information system, a DSS is never fully implemented. The middle-out, iterative, adaptive approach to building a DSS means it is always evolving and changing. New modules are being added, old modules are being revised to add capabilities, revise existing capabilities, respond to changes in policy, etc. The implementation and maintenance process must recognize this situation and must support and facilitate this constant process of change. Sprague and Carlson [1982, p.133] define four types of flexibility that a DSS should have:

1. The ability for the user to confront a problem in a flexible, personal way
2. The ability to modify the Specific DSS so that it can handle a different or expanded set of problems
3. The ability to adapt to new capabilities in dialog, data management, and modeling (i.e., to add capabilities to the DSS Generator)
4. The ability to evolve in response to changes in the technology on which the DSS is based

Flexibility 1 facilitates use of the system and user satisfaction. The other three flexibilities facilitate implementation and maintenance. Flexibility 2 is needed to support the prototyping approach to module development. Flexibilities 3 and 4 make it easy for the SMO to continually improve the system.

One of the big differences between most traditional information systems and a DSS is that updating and maintenance of the modules (in addition to the database) is of critical importance. This includes:

- Refitting parameters (finding new coefficients for the equations)
- Reestimating models (due to changes in the environment)
- Modifying programs in response to changing user needs
- Modifying programs in response to changes in the organization's policies and procedures
- Changing output reports in response to changes in reporting requirements

Murray [1989] describes procedures for updating the equations of one of the models in the EFMS and explains how much effort might be involved. Updating the equations involves four activities:

1. Adding data to the files used to estimate the equations
2. Reestimating the current specifications of the equations
3. Exploring possible respecifications of the equations to exploit the additional data or to accommodate new EFMS needs
4. Testing and evaluating the new versions of the equations

Adding data to the files requires understanding the structures of three large data files and understanding the programs that use these files to create the analysis files. Reestimating the current specifications of the equations requires only understanding the programs that calculate the estimates. However, exploring possible respecifications is more demanding. It requires understanding: (1) the statistical strategy underlying the estimation procedures, (2) the perils for estimation inherent in the available data, (3) the uses to which the loss equations will be put, (4) the programs for calculating estimates, and (5) how to adapt the equations in response to information from the testing and evaluation exercise. Testing and evaluating the new versions of the equations requires understanding: (1) the testing programs, (2) the performance criteria used to evaluate the performance of the equations, and (3) the purposes to which the equations will be put.

This is the process for a single model. Updating all the models in an ODSS requires a considerable amount of work. But this work must be done if the system is to continue to be useful and used. This is only one of the many activities that the SMO must carry out. That is why I recommend that it be a separate organizational entity. Other responsibilities during this phase include:

- Maintaining and updating the database
- Distributing hard-copy reports produced by the system
- Interacting with users (responding to questions, modifying programs, delivering training)
- Interacting with sources of information and technology (external vendors and other departments in the same organization)

CONCLUSIONS

This paper has shown that, although many aspects of a TDSS and ODSS are similar, there are important differences that must be understood if the ODSS is to be successful. Differences in purposes lead to differences in design, differences in managing the development effort, differences in implementation, and differences in maintenance. In fact, some aspects of building an ODSS are more similar to building a traditional information system than to building a TDSS. The suggested approach to building an ODSS combines principles and good practice for building both types of systems.

BIBLIOGRAPHY

Bidgoli, Hossein, *Decision Support Systems: Principles & Practice*, West Publishing Company, St. Paul, MN, 1989.

Brooks, Frederick P., Jr., *The Mythical Man-Month: Essays on Software Engineering*, Addison-Wesley Publishing Co., Reading, MA, 1975.

Carlson, E., "An Approach for Designing Decision Support Systems," Chapter 2 in John L. Bennett (ed.), *Building Decision Support Systems*, Addison-Wesley Publishing Co., Reading, MA, 1983.

John L. Bennett (ed.), *Building Decision Support Systems*, Addison-Wesley Publishing Co., Reading, MA, 1983.

Carter, Grace M., Jan M. Chaiken, Michael P. Murray, Warren E. Walker, *Conceptual Design of an Enlisted Force Management System for the Air Force*, N-2005-AF, The RAND Corporation, Santa Monica, CA, August 1983.

Huber, George P., "Cognitive Style as a Basis for MIS and DSS Designs: Much Ado About Nothing?," *Management Science*, Vol. 29, No. 5, May, 1983, pp. 567-574.

Hurst, E. Gerald, Jr., David N. Ness, Thomas J. Gambino, and Thomas H. Johnson, "Growing DSS: A Flexible, Evolutionary Approach," Chapter 6 in John L. Bennett (ed.), *Building Decision Support Systems*, Addison-Wesley Publishing Co., Reading, MA, 1983.

Jenkins, A. Milton, *Prototyping: A Methodology for the Design and Development of Application Systems*, Discussion Paper #227, Division of Research, School of Business, Indiana University, Bloomington, IN, April 1983.

Keen, Peter G.W., "Adaptive Design for Decision Support Systems," *Data Base*, Vol. 12, Nos. 1 & 2, Fall 1980, pp. 15-25.

Keen, Peter G.W. and Thomas J. Gambino, "Building a Decision Support System: The Mythical Man-Month Revisited," Chapter 7 in John L. Bennett (ed.), *Building Decision Support Systems*, Addison-Wesley Publishing Co., Reading, MA, 1983.

Lucas, H.C., Jr., *The Analysis, Design, and Implementation of Information Systems*, McGraw-Hill, New York, 1985.

Meador, C. Lawrence and Richard A. Mezger, "Selecting an End User Programming Language for DSS Development, *MIS Quarterly*, December 1984, pp. 267-280.

Miller, Louis W. and Norman Katz, "A Model Management System to Support Policy Analysis," *Decision Support Systems*, Vol. 2, No. 1, March 1986, pp.55-63.

Murray, Michael P., *Middle-Term Loss Prediction Models for the Air Force's Enlisted Force Management System: Information for Updating*, N-2764-AF, The RAND Corporation, May 1989.

Reimann, Bernard C. and Allan D. Waren, "User-Oriented Criteria for the Selection of DSS Software," *Communications of the ACM*, Vol. 28, No. 2, February 1985, pp. 166-179.

Relles, Daniel A., *Allocating Research Resources: The Role of a Data Management Core Unit*, N-2383-NICHD, The RAND Corporation, January 1986.

Thierauf, Robert J., *User-Oriented Decision Support Systems: Accent on Problem Finding*, Prentice Hall, Inc., Englewood Cliffs, 1988.

Turban, Efraim, *Decision Support and Expert Systems: Managerial Perspectives*, Macmillan Publishing Company, New York, 1988.

Wagner, G.R., "Decision Support Systems: The Real Substance," *Interfaces*, Vol. 11, No. 2, April 1981, pp. 77-86.

Walker, Robert G., Robert S. Barnhardt, and Warren E. Walker, *Selecting a Decision Support System Generator for the Air Force's Enlisted Force Management System*, R-3474-AF, The RAND Corporation, Santa Monica, CA, December 1986.