NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

ECONOMIC DEVELOPMENT OF SMALL-SCALE INFORMATION SYSTEMS

by

John Louis Ash, Jr.
and
Dale Richard Spaulding

March, 1990

Thesis Advisor: Timothy J. Shimeall

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b. OFFICE SYMBOL (*If applicable*) Code 37	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (*City, State, and ZIP Code*) Monterey, CA 93943-5002	7b. ADDRESS (*City, State, and ZIP Code*) Monterey, CA 93943-5002
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (*If applicable*)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (*City, State, and ZIP Code*)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (*Include Security Classification*)
ECONOMIC DEVELOPMENT OF SMALL-SCALE INFORMATION SYSTEMS (UNCLASSIFIED)

12. PERSONAL AUTHOR(S)
Ash, John, L., Jr. Spaulding, Dale, R.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (*Year, Month, Day*)	15. PAGE COUNT
Masters Thesis	FROM _____ TO _____	March 1990	98

16. SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the US Government.

17. COSATI CODES			18. SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
FIELD	GROUP	SUB-GROUP	
			End-user software development; software development methodologies; computer applications development without professionals

19. ABSTRACT (*Continue on reverse if necessary and identify by block number*)

In virtually any organization, (military or civilian, large or small), there are a number of unsolved small-scale information system problems. They will remain unsolved until we (system analysts and designers) develop a methodology that allows generation of timely and cost-effective automated solutions. The requisite resources exist; they have yet to be combined in the right order for success. It is the primary goal of this research to propose, justify and demonstrate (through a detailed case study) a simple-to-employ methodology for quick and cost-effective development of small-scale information systems (SSIS) without professional analysts, designers or programmers.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL Timothy J. Shimeall	22b. TELEPHONE (*Include Area Code*) (408) 646-2509	22c. OFFICE SYMBOL Code 52SM

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

Unclassified

Approved for public release; distribution is unlimited.

Economic Development of Small-scale Information Systems

by

John L. Ash, Jr.
Lieutenant Commander, United States Navy
B.S., Virginia Polytechnic Institute and State University

and

Dale R. Spaulding
Lieutenant, United States Navy
B.S., Auburn University

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL

March 1990

Author: _____
John L. Ash, Jr.

Author: _____
Dale R. Spaulding

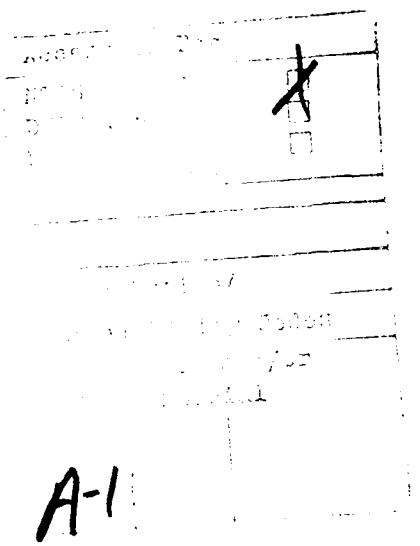
Approved by: _____
Timothy J. Shimeall, Thesis Advisor

_____
Magdi Kamel, Second Reader

_____
David R. Whipple, Chairman
Department of Administrative Sciences

ABSTRACT

In virtually any organization, (military or civilian, large or small), there are a number of unsolved small-scale information system problems. They will remain unsolved until we (system analysts and designers) develop a methodology that allows generation of timely and cost-effective automated solutions. The requisite resources exist; they have yet to be combined in the right order for success. It is the primary goal of this research to propose, justify and demonstrate (through a detailed case study) a simple-to-employ methodology for quick and cost-effective development of small-scale information systems (SSIS) without professional analysts, designers or programmers.



A-1

TABLE OF CONTENTS

I. INTRODUCTION

A. PROBLEM STATEMENT

The current mind set in the development of software applications is biased toward professional system analysts, designers, programmers, large amounts of time (often on the order of years), and complicated methodologies. This mind set often inhibits a good solution to the small-scale information system (SSIS) problem. It breeds an unwillingness to even attempt a solution because of the perceived level of effort required. Taking a step back to gain a clearer view, it is possible that the opposite is true - a significant number of the SSIS that users need (i.e., problems that are worth solving) require only a fraction of this effort. Examples of such applications include the following:

- equipment maintenance records
- parts inventories
- training logs
- health, pay and personnel records

These examples are drawn from the experience of the authors as U.S. Naval officers stationed aboard ship. The problems are well-known SSIS that, if solved

through automation, would greatly increase the efficiency and effectiveness of all concerned. Time management is critical aboard ship and far too many of the above applications and others similar to them are being accomplished manually, thus consuming numerous hours that could be used elsewhere more effectively. For example, the divisional training petty officer spends approximately 6 hours per week on the maintenance of the division's training records. Statistical data requires an additional 5-10 hours to compile depending on the size of his division. An automated SSIS of this type would be simple to develop (given tools already available to the Navy) and would reduce the man-hours required of the divisional training petty officer to man-minutes.

The essence of solving these kinds of applications development problems lies in eliminating the mind set described earlier and replacing it with the idea that, for small applications, the educated user can (to a large degree) develop his own solutions without expensive professional people and time-consuming methods.

In 1982, Martin [Ref. 12] observed the widespread need for SSIS as well as a potential for changing the role of the systems analyst when he said,

> "Systems analysts should advise and encourage the end users in employing new systems. The analysts should assist them, where applicable, in creating their own applications. The role of the systems analyst, then, swings toward being a consultant to end users, thus becoming expert on the facilities for creating applications without programmers."

B. PURPOSE

The 1990's will bring us a to a transition stage in application development methodologies and the technology and tools used in building these applications. End-user applications development is on the rise both because of the interest stimulated by the widespread availability of personal computers and the inability of information centers to keep up with the demand for new applications. The point of transition we enter in can be best described as a gap. This gap exists between the traditional applications development methodologies of the past and the fourth generation languages (4GL's) and code generators that will be common on every personal computer in the near future.

The primary goal in conducting this research is to propose, justify and demonstrate a simple-to-employ methodology for quick and cost-effective development of SSIS without professional analysts, designers or programmers in order to bridge this gap between the traditional methodologies of yesterday and 4GL's of tomorrow. A simultaneous goal is the delivery of a fully functional application for the users in the case study.

C. BACKGROUND

The following highlights, chronologically, some of the popular methodologies that have evolved. Parts of these methodologies are suitable as the basis for the methodology described later. The conclusion of this section describes the selected parts and justifies the selection.

1. Applications Development Methodologies

Early applications development methodologies were proclaimed in the late 1970's which included DeMarco [Ref. 6], and Yourdon's [Ref. 21] structured analysis and structured design methodologies. The requirements and design documents, combined with extensive use of graphic tools, produced a paper model of the system to be and the end user was expected to envision the end product based on this paper model. Once the design document was approved, no changes to this design are permitted. This process, according to Martin [Ref. 12], has "...resulted in the implementation of unsatisfactory systems, especially for those having a moderate degree of uncertainty." Concerning the role of the end-user in traditional approaches to applications development, Connor [Ref. 5] states that, "...user involvement in this structured process was haphazard at best." This was primarily due to the fact that computer based solutions to problems were incomprehensible to the average user.

As personal computers entered the picture in the 1980's, so did a new applications development methodology. The prototyping methodology overcame the inefficiencies of traditional life cycle approaches through active user involvement. This methodology allows users to point to features they like or dislike about the application rather than describing what they think they like through an imaginary paper specification system. Prototyping is an iterative process that is repeated until no new requirements are identified by the user. Studies by Alavi [Ref. 1] have shown that prototyped on-line, user-interactive

systems usually result in a higher level of user satisfaction as well as reduction in the overall cost of an application's development. On the other hand, it has been demonstrated that prototyping from the outset may produce less effective systems due to the purposely reduced emphasis on initial analysis and design [Ref. 4].

In 1985, Burns and Dennis [Ref. 4] combined the advantages of past application development techniques into what they called a "Mixed Methodology". Prototyping is used as a strategy to clarify user requirements during the initial analysis phase. After a working prototype is developed, it provides a basis for writing a set of system specifications. Burns and Dennis [Ref. 4] noted that a mixed methodology is a viable technique, but that it's appropriate only for projects with a high degree of uncertainty and complexity. They state that uncertainty is contingent upon the projects degree of structuredness, how well the users understand the problem, and the developer's proficiency. Furthermore, according to Burns and Dennis [Ref. 4], complexity is contingent upon project size, number of users, volume of new information, and the degree of complexity required to produce the new information.

The object-oriented design methodology was developed in the mid-1980's and seemed to reduce the problems of past applications development techniques. Object-oriented development is simply an approach to software design in which the design of a system is based on the concept of objects or items of importance to the user. Kroenke and Dolan [Ref. 11] further define an object as, "...the representation of an entity in the user's work environment, such as a

customer, salesperson, or inventory item." Identifying these objects is one of the primary goals of the requirements phase of an applications development. After all, these objects and their associated attributes comprise the set of data that is to be stored and manipulated by the application. For example, a customer object will have an associated identification number, name, address and phone number. Once identified, these objects and attributes translate into a system's data requirements. Relationships between them also become apparent.

A case study by Gupta, Rajiv, Cheng, Gupta, Rajesh, Hardonag, and Breuer [Ref. 8] illustrates that the object-oriented methodology supports rapid prototyping, letting developers gradually refine a set of objects instead of specifying and developing a new and complete application. Future applications are then built from libraries of standard objects or subassemblies that can be spliced together to fulfill the users' specification much faster than traditional procedural approaches.

The above methodologies have produced many systems that have had an impact in our society. Computer literacy in the United States is on a rapid rise. For example, as Admiral Tobin [Ref. 18] stated in his September 1989 address to computer technology students at the Naval Postgraduate School, "Ninety percent of the entering freshman class at the United States Naval Academy were knowledgeable in the use of a personal computer and the MS-DOS operating system." It is the goal of this thesis to demonstrate a

methodology for true end-user development of small-scale informations systems (SSIS).

2. Conclusion

The object-oriented approach is the basis of the methodology we propose and demonstrate through the case study presented in the next chapter. The first phase in the SSIS development process is determining functional requirements. Dataflow diagrams prove useful during this phase and can be used to diagram both the current and proposed systems. During the diagramming process, the objects are identified and refined. Actions on these objects such as create, edit, delete and display can also be represented using this graphic technique.

Once the objects are identified, they must be described in detail for further use in the SSIS building phase. The object-oriented methodology provides tools such as object diagrams, object definitions and domain definitions to provide this descriptive detail.

Various tools of the structured design methodology can be used in the design phase of SSIS development. In particular, the structure chart is valuable when used as an aid to refine a diagramatic overview of the system (i.e., a Data Flow Diagram) into a detailed plan which depicts all operations (functional modules) and the data flows between them. Once completed, the structure chart allows the developer to see "the big picture"; how all the pieces or modules fit together and interact to form the SSIS.

Prototyping is an effective means to clear up uncertainties of the design phase while, simultaneously, watching the final product unfold. Prototyping is the backbone of the SSIS building phase. The functional modules identified in the design phase are constructed or obtained from a library and sewn together during the building phase. The end-user can experiment with different screen displays and other features such as function key use and color patterns, firmly identifying what works best for him. The process is an iterative one that continues until the end-user is satisfied with the product. If the development process has gone well, the prototype evolves into a deliverable final product; a product which fully satisfies the end-user's requirements because he was an integral part of its development.

Prototyping provides one other key advantage- speed. Improving (increasing) the speed at which applications are developed increases productivity and decreases development costs (particularly in manpower and dollar resources). Coupled to the issues of improving speed and productivity is the requirement to maintain product quality. This poses some difficult control problems for the manager. These problems are addressed briefly below and in greater detail in Chapter 3.

The above tools and methodologies provide the basis for a simple-to-employ technique for quick and cost-effective development of SSIS's. These techniques will be applied in the next chapter through an actual SSIS case study.

D. RESEARCH QUESTIONS

1. Of the existing methodologies, are any suitable for true end-user applications development? What are the current trends toward achieving the goal of true end-user SSIS development?

There are a number of common threads linking all small-scale information system applications. First, each deals with a finite set of objects, or items of interest to the user. Next, each of these objects has a relatively small and stable number of associated attributes that are of concern to the user. Finally, there are a limited number of common operations that are performed on instances of these objects. These "usual" operations are: add, delete, update, query and display.

Given these commonalities, it logically follows that a generic methodology is attainable for use in development of these types of applications. Furthermore, there are tried and tested modules of code in a variety of programming languages that can be used to perform all the generic operations listed above. Applications development should use these modules not only because they relieve the developer of the programming process, but also because of the significant cost and time savings that can be realized. The debugging process is simpler when you are working with modules that are certified to perform as advertised. Finally, SSIS development by end-users is achievable when re-useable modules are employed effectively.

The object-oriented design methodology fits well into the scheme of end-user applications development. After all, the user is the one most familiar with the objects in his environment. Once a library of "usual" operations modules is made available to the end-user either through purchase or in-house development, SSIS applications can be built quickly (on the order of 30-90 days) and inexpensively (under $1000.00) by "sewing" together the object(s) of interest in the users' environment with these re-useable modules of code (which do the "usual" things with instances of the objects) using the tool kit described below. Rapid prototyping is easy this way, allowing an iterative design process which focuses on end-users as an integral part of the development process- a process which ultimately "grows" the finished product as the user envisioned it. This process can be understood and employed by end-users without the aid of systems development professionals.

The roles of the programmer, analyst, and user have changed rapidly over the last 15 years. In the late 1960's most computer based solutions to problems were solved through the sole efforts of the programmer. By the late 1970's, applications were being developed by a team consisting of the systems analyst and the programmer. This trend continued into the 1980's, when the user became a part of the applications development process. As re-useable modules for a wide variety of applications appeared on the market in the mid-1980's, systems analysts relied less and less on the efforts of programmers in the development of their applications. A logical next step in this evolutionary

continuum would seem to be for the end-user to begin developing small-scale applications without professional analysts or programmers. Then, as Martin [Ref. 12] proposed in 1982, the analyst can become expert on the facilities for applications development without programmers and introduce these facilities to end-users.

Among the more recent work is Meier's efforts in Regina (Saskatchewan, Canada) [Ref. 19] which best characterizes the current state of end-user applications development technology. Meier, city of Regina's Director of Information Systems has taken bold steps in educating his users and encouraging them to develop their own applications. In his own words Meier wants

> "...end-users to understand our terminology and symbology - things like entity-relationship (ER) diagrams. We have spent time front-loading the users with knowledge, and I think it's paid off."

Meier feels that end-users will become self-sufficient enough to design and build their own applications. His group has developed its own 25-module applications generator that virtually eliminates coding, and his end users will use it in developing their own applications.

Through Meier's work it can be seen that end-user applications development without professional programmers is possible and is at least being attempted. However, until the gap between past and future methodologies is closed, the need remains for a methodology that allows the end-user to develop SSIS's today.

2. How may the hypothesis that true end-user applications development is now possible be supported?

A simple-to-employ methodology will be demonstrated and justified through a case study of an actual SSIS (Flight Physical System) that was developed by the authors for the U.S. Army Flight Surgeons at Fort Ord, CA. Details of this study are provided in the next chapter.

3. What software and hardware tools are useful in developing simple applications?

The software development workstation (tool kit) used in the case study consisted of the hardware and software tools listed in Figure 1.1.

4. What baseline skills are needed?

A basic working knowledge of the applications development process is needed by the user if he is to create his own applications. This is not to imply that one must complete the intensive training requirements of a systems analyst; simply talking to an analyst or doing some reading on the subject at the library will give the requisite background on the how the applications development process works.

The user must also be familiar with his computer system, its operating system, and a good programmer's text editor or word processor. The text editor is a vital part of software development in that any new source code or any tailoring/modification to the source code of the "tried and tested" modules

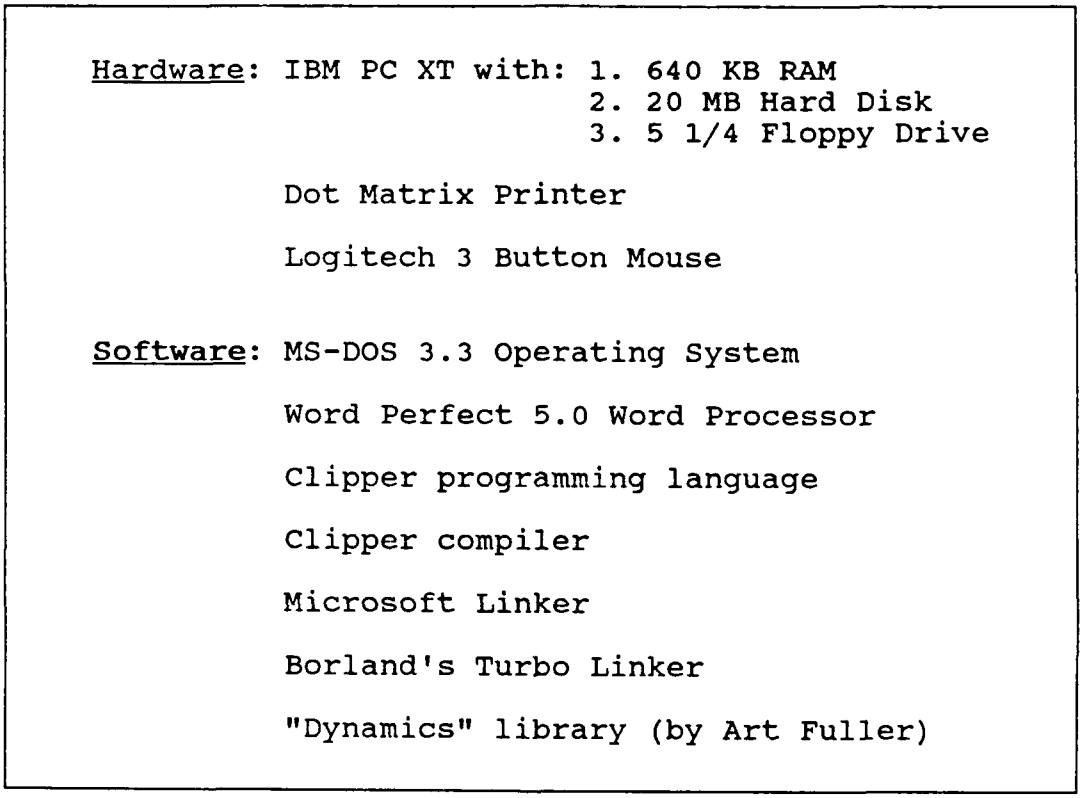


Figure 1.1: Software Development Workstation

will be accomplished using this text editor. Work in this area accounts for a significant amount of the time used to develop applications.

Other helpful resources include references on systems analysis and design, database application development, and structured programming. These references are not required for successful SSIS development by end-users, but they do aid the end-user who is not familiar with the software development process. Manuals and reference guides for the particular development environment used are a mandatory (e.g., Clipper and other programming languages, text editors, automatic code generators and documenters, etc.).

Finally, experience with any structured programming language is helpful. Prior to undertaking this research, the authors completed programming courses in both BASIC (5 wks * 3 hrs/wk = 15 hrs of instruction) and Pascal (10 wks * 4 hrs/wk = 40 hrs of instruction). The end-user SSIS developer may also find comfort in having professional analysts and programmers just a phone call away.

5. What is management's role in end-user applications development?

Stoner and Wankel [Ref. 17] state that, "Management has been defined as the process of planning, organizing, leading, and controlling the efforts of organization members and of using all other organizational resources to achieve stated organizational goals." This definition is also useful in defining management roles in end-user applications development.

a. **Planning**

As with any new task or process, managers must think through their goals and actions in advance. The manager must ask questions such as: What does my organization wish to attain through end-user applications development? How will this goal be attained?

b. **Organizing**

The effectiveness of any software development technique depends on the manager's ability to harness his resources. Coordination between the end-user developer and professional analysts and programmers of an IS staff is a must. Promoting this coordination is part of the manager's role.

c. **Leading**

The manager must establish the proper atmosphere and attitudes in order to achieve organization goals. End-users need to be given the time and incentive to develop quality applications that will improve productivity for themselves as well as the organization as a whole.

d. **Controlling**

In controlling the efforts of end-user applications development, the manager must schedule and conduct formal and informal reviews to track the progress of the developer and ensure that his efforts are in keeping with the organization's goals and directives. Time management becomes a critical factor because this sort of applications development will generally not

be the user's primary job. Finally, the manager must guard against "re-inventing the wheel". Time is a valuable resource; managers must develop and manage an internal library of modules as an investment that will save time and improve quality of future applications development projects. Once end-user software development takes hold, the proliferation of applications which follows highlights the necessity to establish controls to manage and use them effectively as shared resources within the organization.

The next chapter begins with an overview of the FPS case study. Following the overview is a detailed description of the SSIS development methodology that was derived and employed during the FPS case study. Chapter III discusses SSIS development successes and hard spots discovered during the case study effort and provides recommendations for improvements. SSIS development managerial issues and case study development statistics are also included.

II. FPS CASE STUDY

A. OVERVIEW

The body of this chapter is devoted to describing the methodology that was evolved and employed in the development of an SSIS for the Army flight surgeons at Fort Ord Aviation Medicine Clinic in Fort Ord, California. The case study application is entitled "Flight Physical System" (FPS), a title coined and assigned to the project by the authors. Highlighted are the "how's" (tools, process, methods) and the "why's" (reasoning) of what was done in the course of the SSIS development.

The definition of SSIS implies that the number of developers is very small. The SSIS development process is geared toward an environment where the end-user(s) and the developer(s) are the same individual(s) or work closely with each other in the discharge of their regular duties. As such, all players are well familiar with requirements for the SSIS, so the question of "what" to build is much less critical than the question of "how" to build it.

It is possible to apply the SSIS development process when the developer and user are not one and the same. This complicates the process, primarily in the areas of requirements analysis and direct user involvement, as described in

Chapter III. Such complications require effort beyond that expected of the developer who develops only for himself. For practical purposes, since the process succeeded under the latter, more complicated situation, they can reasonably be assumed effective under the former's less restrictive conditions.

After researching the methodologies discussed in the first chapter and examining the advantages and disadvantages of each, the SSIS development process was formulated by a combination of methods well-suited to small scale development. During the case study, these methods were re-evaluated, modifying those that almost worked and replacing or abandoning those that didn't work at all. This evaluation, modification and replacement was based on developments described in the open literature and on discussions of progress with the application's users as well as with faculty advisors and peers.

A successful working prototype and associated documentation was completed and delivered to the group of users. The prototype implements all basic functionality prescribed by the users in the original project definition. A number of new or expanded requirements evolved in the course of development (e.g., additional reports desired, an automated appointments scheduler/manager, etc.). The majority of these were incorporated into the design but, due to project scope limits, many of the emergent ("nice to have") requirements were not implemented in the product delivered. Comprehensive documentation and loosely-coupled modularity of the design permit the controlled addition of these features at any time in the future.

B. FPS BACKGROUND

Involvement with the FPS project began with a request for automated system development assistance by the Fort Ord flight surgeons to the Information Systems faculty at the Naval Postgraduate School (NPS). The flight surgeons' effort to automate some of the routine functions of their existing flight physical reporting operation was initiated by a letter from the Army Aeromedical Activity (AAMA) in Fort Rucker, AL, an agency which supervises the 53 Army flight surgeon clinics worldwide [Ref. 10]. The AAMA's interest in automating certain flight physical functions was to reduce the cost and administrative burden of processing the large amount of paperwork generated under the current manual system. Its letter solicited suggestions or automated solutions from the staffs of the clinics.

The NPS IS faculty offered the Fort Ord surgeons' request as a potential thesis topic. After a preliminary feasibility study (discussed below), the authors determined that FPS represented a typical SSIS and would fit well as the development project in a case study. It was also attractive to the extent that the resultant system (both the developed application and the findings of this research) was of potential use in fulfilling current needs in the military.

The remaining discussion in this section will give the reader an appreciation for the situation/problem that FPS was intended to address. Though details of FPS itself are not important, such an appreciation is essential to the full

understanding of the methodology discussed in subsequent sections, which is given in the context of the FPS development process.

The three flight surgeons assigned to the Fort Ord Aviation Medical Clinic are responsible for maintaining records and conducting annual flight physicals for approximately 1500 military aircrewman assigned within their area of responsibility. For each aircrewman's annual physical, there are two medical forms generated to document the examination's results and the examining physician's recommendation regarding each aircrewman's medical eligibility for flight duty.

These forms must be forwarded to the AAMA where they are reviewed for completeness and accuracy by other flight surgeons who make the final determination of each aircrewman's medical eligibility for flight duty. After review, each set of forms is manually keyed into a central computer database at the AAMA by data entry clerks.

The system in force at the Fort Ord clinic for flight physical administration is fully manual. Each set of forms for each aircrewman is re-generated and processed for each annual flight physical. The clinic's flight physical database is in the form of some 1500 individual medical records for the aircrewman served. The database is maintained by a small staff of civilian and Army enlisted personnel. A large part of the physicians' time is spent in processing and handling the medical exam forms (minor calculations, identifying exam results

that are out of prescribed limits, re-generating mandatory standard comments on the forms, filling in standard entries, checking accuracy and completeness, etc.).

In spite of the significant time and effort dedicated to ensuring complete and accurate exam forms, more than 8000 forms are returned by the AAMA annually to the various clinics for correction and re-submission [Ref. 10]. This results in excessive postage costs, significant flight surgeon time wasted in administrative re-work (largely due to minor errors and omission of mandatory data in the forms) and inordinate delays in final determination of affected aircrewmen's eligibility for flight duty.

The core goal of FPS was to reduce the administrative overhead, error rate and cost of processing flight physical examination data by automating and replacing portions of the existing manual system. Beyond these core goals, there were other benefits which could be derived from the system envisioned. These include the ability to automatically generate and maintain a tickler file of aircrewman due for flight physical (not feasible using the current system), the ability to forward aircrewmans' aeromedical histories by disk or modem, improved readability of typed machine-generated forms over forms hand scribed by physicians (not an insignificant benefit), the potential to eliminate hardcopy forms altogether by forwarding exam data to the AAMA electronically and the associated potential for reduction or elimination of AAMA data transcribers (a savings of $150,000/year for a 50% reduction) [Ref. 10].

C. PREPARING FOR DEVELOPMENT

As touched upon earlier, there are some preliminary steps recommended for the new or novice SSIS developer that will pay dividends in productivity on the long term. These steps include doing background reading and talking to software development professionals. Formal courses of instruction in structured computer programming and/or software development or software engineering techniques are another avenue of preparation.

1. Background Readings

The writings of Martin [Ref. 12] (end-user development), Yourdon, Constantine [Ref. 21] and Page-Jones [Ref. 15] (structured design) will give the new developer a general appreciation of modern and traditional software development terms and the scope of what software development encompasses. Computer and software development magazines are an excellent source of reading material on the latest advances in the software development arena. Figure 3.1 in Chapter III includes examples of the open literature useful in preparing for SSIS development.

2. Professional Consultants

Talking to software development professionals can be beneficial in a number of ways. First, points of contact are generated should any assistance become necessary later on in the development process. An experienced developer can likely recommend the best and latest software development tools

for a particular application, saving time and money in selecting and building the development environment. He can also be a good source of reference material for further reading and preparation prior to starting a project. Finally, inexpensive software may already exist that could be tailored to suit project needs; the software development professional will likely be aware of this and save both time and money by avoiding "re-inventing wheels".

3. Formal Courses of Instruction

This level of preparation effort is somewhat extreme in the context of non-professional simple systems development. Formal education starts to violate the boundary that separates the casual user-developer without a systems development background and the schooled individual with knowledge and experience normally aquired only by those pursuing a career related to systems development. However, for the organization planning a continuing regime of user-developer SSIS efforts or for the individual considering becoming a full-time SSIS developer, the effort may be justified in order to establish a level of developer proficiency and productivity commensurate with applications development goals.

If formal instruction is considered feasible, courses in structured programming, software development and software engineering are recommended (in that order). Instruction in structured programming is considered to be of greater benefit than the latter two if all three are not possible. If all three are possible, it is recommended that they be taken in the order given due to the

cumulative nature of each (each tends to build on or assume knowledge of material presented in the previous course). Such courses are available at local colleges and universities and may be offered through Navy programs (consult NARDAC, NAVDAC or local facilities' IS departments for information).

Throughout development of FPS, the development team continued with graduate courses in information systems at the Naval Postgraduate School in Monterey, CA. Prior to commencing the project, both team members had taken approximately 55 classroom hours each of structured programming (BASIC and PASCAL) and 36 classroom hours each of structured software analysis and design training. These courses (and their associated readings) provided a foundation to build from in the application development task, but are not considered mandatory to success in the development of SSIS.

D. FEASIBILITY STUDY

Before expending any effort toward development of an SSIS, a feasibility study should be completed. A careful study will assess the risks involved with the project and provide developers and managers the information required to determine whether or not to continue. The study can be detailed or general, depending on what managers/developers are comfortable with in making their decision. The initial FPS feasibility study is included in the Appendix. The basic format is distilled and tailored for FPS (SSIS) from examples given by Whitten,

Bentley and Ho in their systems analysis and design text [Ref. 20]. As a minimum, the SSIS feasibility study should address the following:

- Project Scope: A statement of what organizational need(s) the SSIS will satisfy. A description of what end products will be generated should be included.

- Project Constraints: A list of limitations and events which constrain the SSIS' development. Such items as schedule requirements, developer ability, organization's need for the application, and available resources (time, money, materials, etc.) that dictate how the project will develop are included here.

- Alternative Solutions: A list of all possible alternatives to continuing with the project as currently envisioned. From these possible alternatives, managers/developers can distill those that are feasible, and ultimately arrive at a course of action.

- Cost/Benefit Analysis: A study of estimated resources required to develop and implement the SSIS versus the benefit to be derived from it once it's in place. Benefits need not be of a dollar value; savings in effort, hours, complexity, worker fatigue, etc. may be metrics used in measuring the benefit to the organization.

- Resources Available: A list of the manpower, material, computer, financial, expert and reference resources available for manager/developer use in weighing against estimated resources required.

- Recommendations: A statement of the conclusions and decisions drawn from the study. This information serves as the initial documentation of the earliest development actions and the reasoning behind them. Clear documentation of decisions made is of key importance to original developers later when trying to re-construct and understand how the application evolved to its current state. It is also essential when future others are trying to maintain the application (correcting errors in or trying to modify or update the application).

Accurately estimating the size, time to complete, cost and effort required to deliver a project is a complicated task and is the subject of many volumes of software estimation literature. The steps suggested in section C above will better

prepare the new or novice SSIS developer to take on the uncertainty of estimation for his project. Accumulated experience will allow greater estimation accuracy.

Project feasibility should be reviewed periodically until the project is completed. If at any point the costs are discovered to outweigh benefits, termination of the project should be seriously considered.

E. REQUIREMENTS ANALYSIS

1. SSIS Documentation

Good documentation is of key importance in any future effort to correct, change or update an application. It also proves useful at later stages in the development process itself, especially when working with a lengthy development timeline.

Documentation is essentially a record of what was done in the course of SSIS development, with a supporting discussion of why and how it was done. It also includes commentary in an application's programmed source code explaining what function that code performs as well as any text or operator manuals the user will need to fully and properly run the application. Much of this may appear obvious or insignificant for the purposes of recording, but experience (both casual and professional) shows that there are few details that aren't worthy of comment in some way. Documentation begins at the immediate

onset of a new project, and continues through the day the project is determined to be complete.

As a general rule, an application cannot be "over-documented". It is much simpler to line out extra commentary than it is to try to accurately recall what decisions and actions occurred a week or a month ago. Notes and comments were found to be of great reference value in all stages of FPS development. Among other things, they served as "minutes" of previous meetings between team members, advisors and users, as reminders of ideas conceived early on but implemented later, as the foundation for building the FPS user manual and as logic outlines for modules of code to be built into FPS.

The finished documentation of the FPS development process that accompanied the application when it was delivered to the user was distilled from notes, kept in diary form, over the six month period in which FPS was developed. The FPS diary (see diary excerpts, Figure 2.1) contained a chronological record of all activity related to the project. Included were thoughts, remarks and ideas generated by team members, advisors, and users along the way. Additionally, all iterations of diagrams, write-ups, source code and notes were dated and filed for future use. These, with notes scribbled on them, were helpful later in deducing how and why certain development decisions were made. From the finished documentation, the users can trace and understand the logic and process which developed FPS, learn and properly use all its features, and perform any required

APRIL 1989

4/11 Proposed dual thesis; case study to be FPS.

4/11 Called **Dr. A. USER, Cpt, USA: 123-4567/68** (SPONSOR);mtg scheduled for 1430, 12 April 89.

4/12 Briefed A. USER, toured clinic, got name-/nr of Fort Rucker **(AAMA) POC: A. CONTACT, Maj, USA: A/V 123-4567** (IS type; not a doctor). **Silas B. Hayes POC: A. DOCK, Major, USA: 123-4567** (IS Dept Head-). User briefed team on general operations of the clinic and discussed what he wanted to do with the project.

4/12 The system must serve 1500 aircrewmen (who have their annual flight physicals at the Fort Ord clinic).

4/12 Physicals are conducted annually in aircrewman's birth month or w/in 90 days prior to his birth date (process roughly 125 flight physicals per month).

4/12 User wants to use DBase III+ since it's funded and available to him. (Software constraint).

4/12 User can get one Zenith 248 PC w/ color monitor, dual 5.25 inch floppy drives and one hard drive of unspecified capacity. Says he can probably get a laser printer. (Hardware constraint).

Figure 2.1: FPS Diary Excerpts

maintenance in the future (future corrections, changes or updates to the delivered version of the SSIS).

2. Identifying Requirements

This step is to describe detailed goals for an SSIS. The bulk of the requirements are already established if a feasibility study has been completed. A precise and prioritized list of what the SSIS will do is the product of the requirements identification effort.

For the user-developer, identifying requirements is simpler than the traditional situation where the developer must learn the user's work environment and then try to accurately understand the user's lay-explanation of his requirements. The user-developer already understands his work environment and needs only to decide what he wants of his SSIS. There is little guesswork involved; the developer knows best what he wants for himself.

A brainstorming approach is suggested for making the initial list of SSIS requirements. All functions that are considered mandatory in the SSIS product should be written down. The entries should be of a form similar to "The SSIS **shall** do thing x", for each of these mandatory functions. The statements should be clear and concise, and include supplementary remarks detailing why the functions listed are considered mandatory. Also, any thoughts and discussions which may be helpful later during the design and implementation phases should be recorded. The same procedure can be used to list other, non-critical functions that may be optionally included in the SSIS (e.g., "The SSIS **may** do thing y").

The length of the list is not important. The important thing is that it be as comprehensive as possible. A sufficient list details every function the SSIS could be expected to provide.

When the list is done, it should be reviewed carefully. Shuffling the entries around, discussing them with others concerned, and prioritizing them will help in paring the list down to only the essential requirements. Prioritization of non-critical (optional) requirements will help guide additional development (in the form of enhancements) after the essential functionalities have been provided for in the detailed design and implementation phase. The SSIS is not limited to the functions derived in this list; functionality may be added at any time in the development process. Functionality can added without penalty provided development proceeds in the controlled manner of the stepwise methodology suggested here. It may also become evident later that some of the requirements listed need modification or are altogether inappropriate and must be deleted.

Once the refined list of requirements established, the functional area of concern to the SSIS is fenced off and it is possible to continue with requirements analysis.

3. Diagramming the System

One graphic tool commonly used in the structured methodologies is particularly useful in visualizing the overall SSIS. The data flow diagram (DFD) gives a high level, "big picture" snapshot of the entire system at an early stage in development. It can be constructed from general requirements and data

information; only minimum analysis is required prior to generating the first iteration.

a. The Data Flow Diagram (DFD)

Figure 2.2 contains one of the earliest versions of the FPS DFD; Figure 2.3 contains the finished version. The constructs used to build them generally follow those given by Page-Jones in his structured systems design text [Ref 15]. The particular symbology used is not important; the developer is free to represent the information in the form that is most meaningful to him. Since others may refer to documentation during maintenance, simplicity and consistency in symbology are recommended. What is important is the high level system information conveyed through these powerful graphic tools.

The highest level (or "context") DFD displays the entire SSIS, the essential data flows into and through and out of the SSIS, and the sources and recipients of SSIS data and products. There are no direct temporal or sequence specifications. There are no references to specifics of implementation; how the system will do what it does is not a concern at this high level. The context DFD shows the active components of the SSIS (including people) and the data interfaces between. With this sort of information, the developer can begin to successively break down the components into lower levels of greater detail until a full understanding of the SSIS and its requirements is achieved.

In the traditional structured methodologies, the context DFD is normally broken down into progressively more detailed diagrams. In the

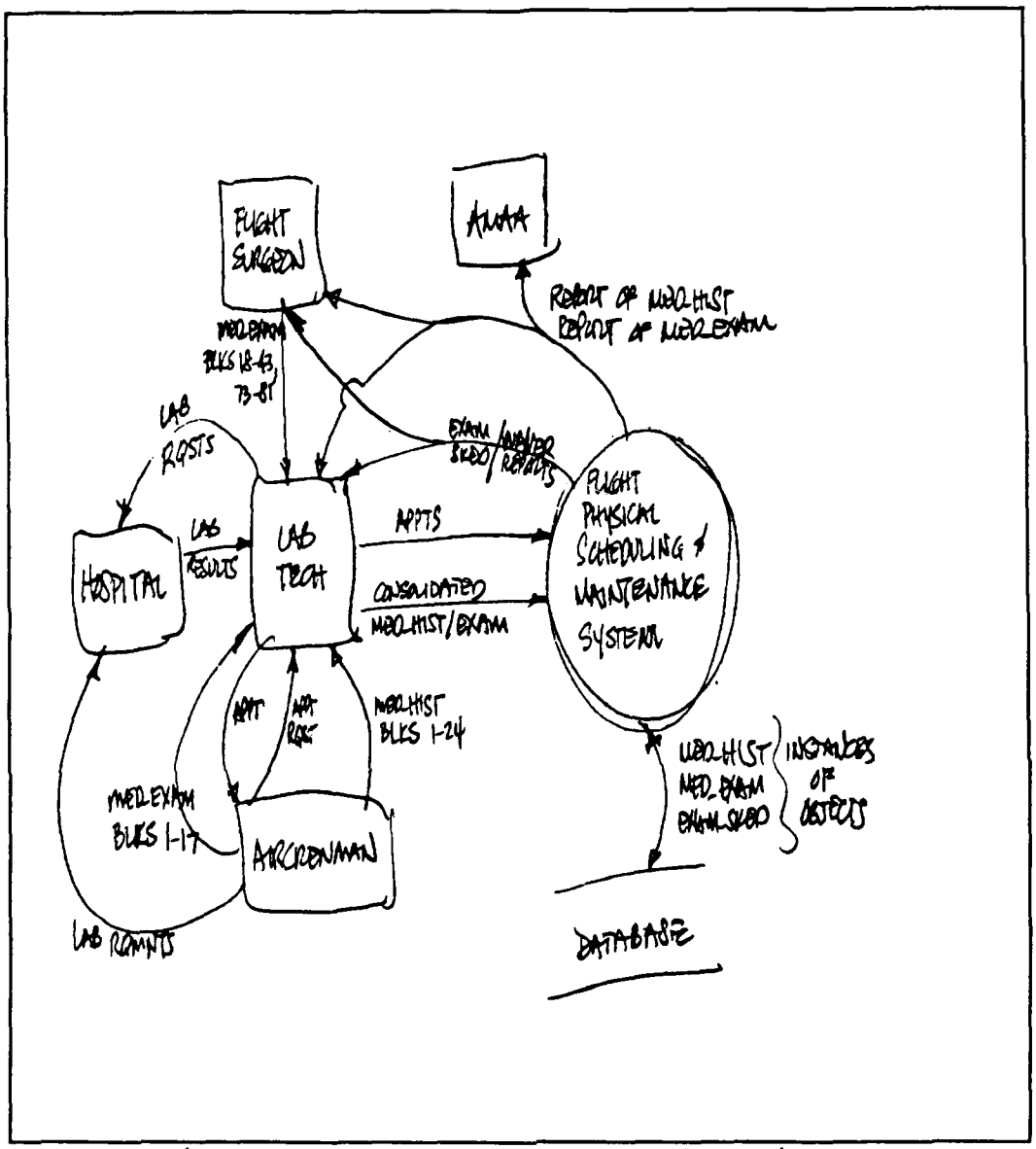


Figure 2.2: Early FPS Data Flow Diagram

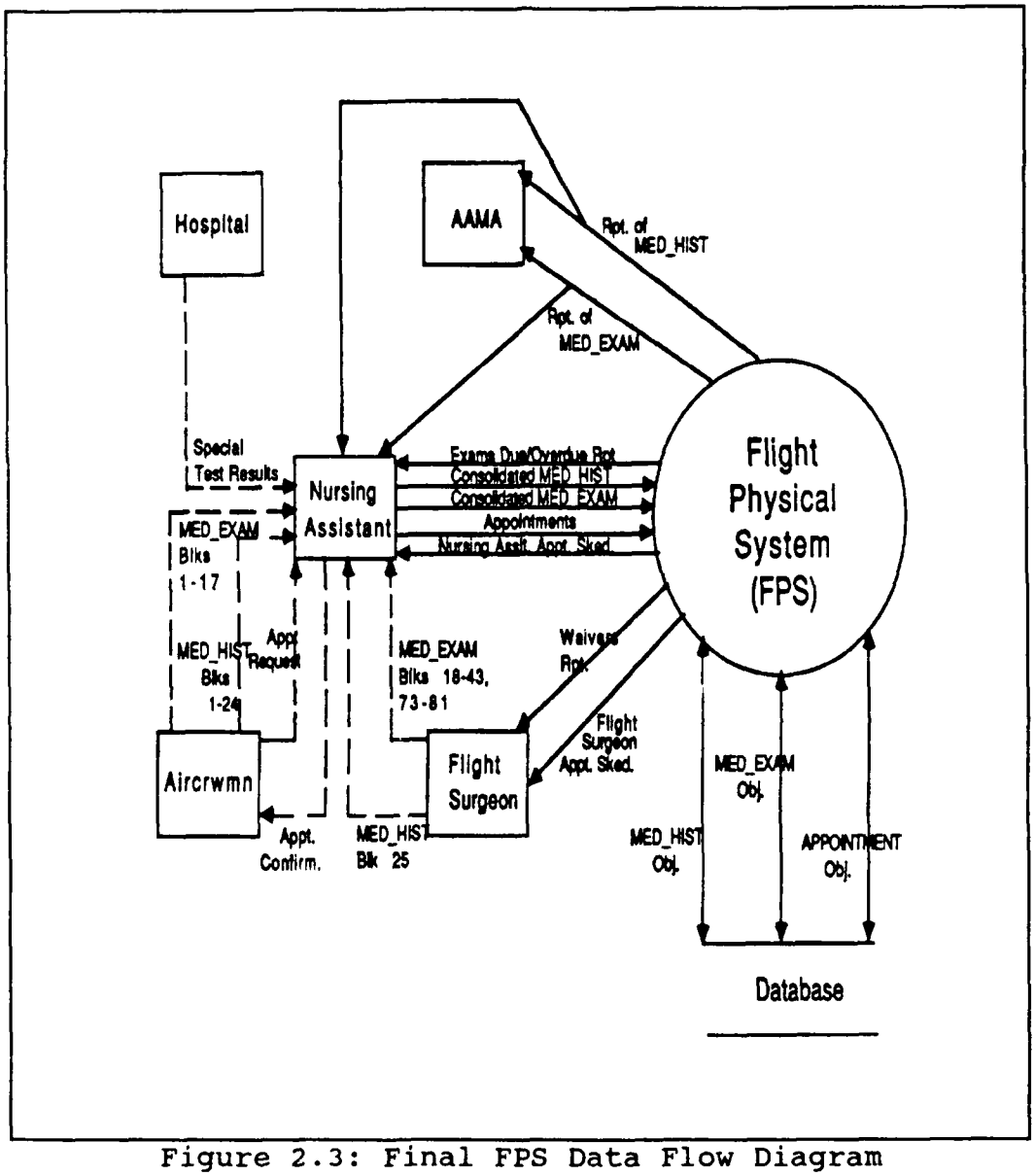


Figure 2.3: Final FPS Data Flow Diagram

development of small systems, the highest level DFD usually yields the information needed to continue with the development process. If the developer feels that more detailed data flow diagramming would help to better understand the system at this level, then more detailed diagrams should be generated and studied.

Once a DFD is generated, it should be inspected to be sure it is free of basic errors. For example, each process must have at least one input and one output; no process can divine output from thin air and a process that produces no output from its inputs does nothing and needn't be there in the first place! The diagram should accurately represent all components necessary for the SSIS project.

In a project of any complexity, the DFD will almost certainly change several times by completion of the design phase. This is no cause for alarm; it is constructed/re-constructed with relative ease. As such, expectations of a perfect product on the first go round are unfounded. More appropriate are quick pencil sketches that can be refined through the remainder of the development process. Holding this perspective will help keep the developer from mistakenly becoming overly involved in or attached to early diagram iterations. This is a mistake that might commit the developer to premature design decisions that lead to inferior SSIS products. Once it has provided the high level information sought during analysis, the DFD should be set aside for later reference and, ultimately, inclusion with system documentation.

With FPS, pencil and paper diagrams were used until the DFD was finalized, then an Apple Macintosh computer and graphics software were used to re-create it in the prettier form shown in Figure 2.3. There are software packages available commercially that automate these sorts of graphic tools and produce fancy output. These products are not essential to SSIS development but may prove to be convenient and/or labor-saving additions to some developers' tool kits.

4. Objects and Data Requirements

So far, the development process draws from elements of the traditional, structured methodologies. At this point in the analysis, the object-oriented methodology becomes useful for its tools which aid in identifying elements from the user's environment that must be represented and manipulated in an SSIS. These elements, referred to as "objects", have specific attributes which must be identified in order to fully describe them. There are also specific relationships between objects (through related or shared attributes) that should be documented and considered before proceeding further. For user-developers, this process is simplified by avoiding the barriers posed by users trying to communicate thoughts to developers.

Guidance for developing FPS objects was taken from Kroenke and Dolan's database development text [Ref. 11]. Since FPS is a database SSIS, this text proved particularly relevant. Any good text on object-oriented design should provide a good stepwise tutorial for creating objects. There will inevitably be

differences between authors in the terminology used and the exact steps prescribed in building objects, but the basic principles and end results are largely the same. The following summarizes how the objects for FPS were developed.

Since the FPS developers were not user-developers, additional effort was required to learn and understand the user environment (largely in the form of user interviews and observation of the flight physical process). The DFD was updated several times during this period to reflect what was learned. Once the flight physical process, its components, and its dialogue were understood, the list of potential objects was boiled down to four final objects. A sampling of these objects (in diagram form) appears in Figure 2.4. The user-developer saves time, effort and reduces margin for error during this phase since he is a already part of and understands the existing process.

The MED_HIST and MED_EXAM objects are derived from the two existing forms used to document results and data gathered during a flight physical. The AIRCREWMAN object is the common thread running between all objects, and represents personal data that must be gathered for each individual that takes a flight physical. The APPOINTMENT object represents information required to schedule and manage the office appointment book. The names of the objects are in a form that is abbreviated to conform to the rules of the programming language used in FPS implementation. Collectively, these four objects encompass the data relevant to the functions indicated in the

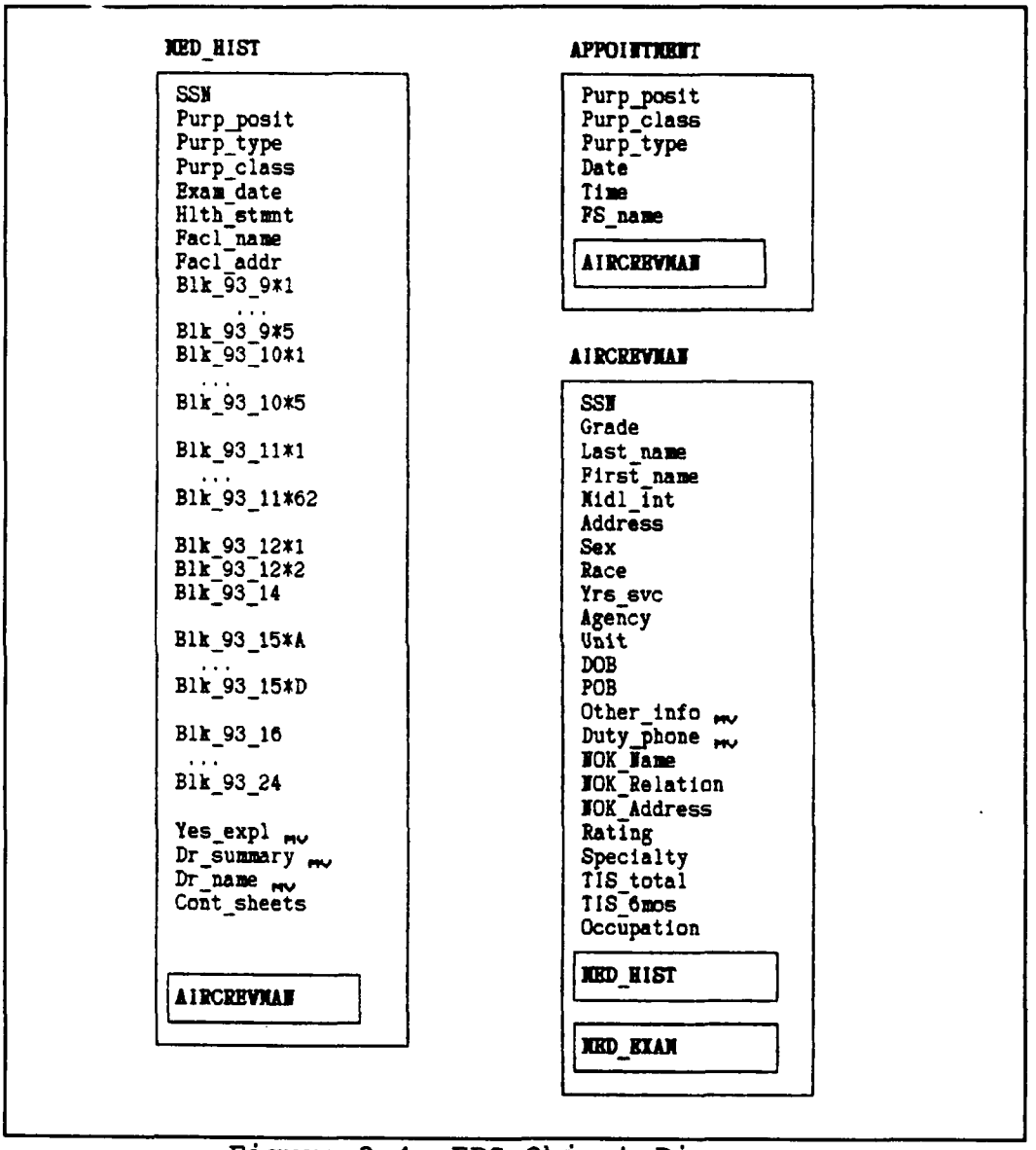


Figure 2.4: FPS Object Diagrams

requirements list. They are also adequate to support the automated processes detailed in the data flow diagram.

With the objects identified, the associated properties or "attributes" which fully described them could be listed under each. For example, each instance of AIRCREWMAN has a social security number (SSN), first and last names, an address, and so on. An instance of an object is a particular occurrence of that object in the user's environment. For example, Captain Joe John Jones, U.S. ARMY, 123-45-6789 is an instance of the AIRCREWMAN object. In total, FPS users at Fort Ord have about 1500 instances of the AIRCREWMAN object, since they conduct flight physicals for about that many individuals in their area of responsibility.

The attributes for the MED_HIST and MED_EXAM objects were taken from the blocks that appear on each existing form, less those items that were covered in other objects. For example, both forms require the examinee's name and grade. These items are included among the AIRCREWMAN object's attributes and should not be duplicated. Each object requires an attribute (or group of attributes) which uniquely identifies each instance of it from all other instances of that object. This attribute is referred to as its "key". In the MED_HIST, MED_EXAM and AIRCREWMAN objects, the key attribute "SSN" serves to uniquely identify each instance of each object. In the APPOINTMENTS object, the attributes "Time" and "Date", in combination, uniquely identify each instance of APPOINTMENT.

It should be noted that the "SSN" attribute is duplicated in three of the four objects. This establishes a relationship that ties together the data contained in separate objects for each individual. The AIRCREWMAN object also appears as an attribute in each of the other three objects. This establishes other relationships that integrate related objects in the FPS database. These relationships are necessary and important to manipulating and displaying FPS data when the users operate the system.

The attributes subscripted "MV" (for "multivalued") can have single or multiple values. All non-subscripted attributes take only single values. For example, the "Duty_phone" attribute in the AIRCREWMAN object is multivalued since an aircrewman may have one or several duty phone numbers. On the other hand, the "Grade" attribute can have only a single value which corresponds to the grade or rank of the aircrewman.

Once all objects and their associated attributes are identified for an SSIS project, more details about them can be specified. These details are documented in the form of domain and object definitions. Figures 2.5 and 2.6, taken from FPS, show sample domain and object definitions, and illustrate the following discussion.

a. Domain Definitions

A domain definition fully describes the complete set of values an attribute may take on. These definitions make up an important part of the documentation that is used by both the developer and others in maintaining

```
Accommodation
 Text 4
 Not required IAW the Administrative Guide for
 the Army Flight Surgeon - 1989

Address
 Text 120, mask:
           Organization Name (Text 40)
           Number and Street Name (Text 40)
           City, State and Zip code (Text 40)
 An individual's or an organization's mailing
address

Agency
 Text 30
 Name of agency/unit an aircrewman is a member
 of/assigned to

Albumin
 Text 8
 Albumin test results on urine sample
(positive or negative)

Audiometry
 Text 2
 Audiometry results using puretone audiometric
 procedures

BloodType/RH
 Text 6
 Blood type/RH factor of examine

Body_fat
 Numeric 2, Mask NN
      where NN is % body fat from AR 600-9,
  table 1
```

Figure 2.5: Sample FPS Domain Definitions

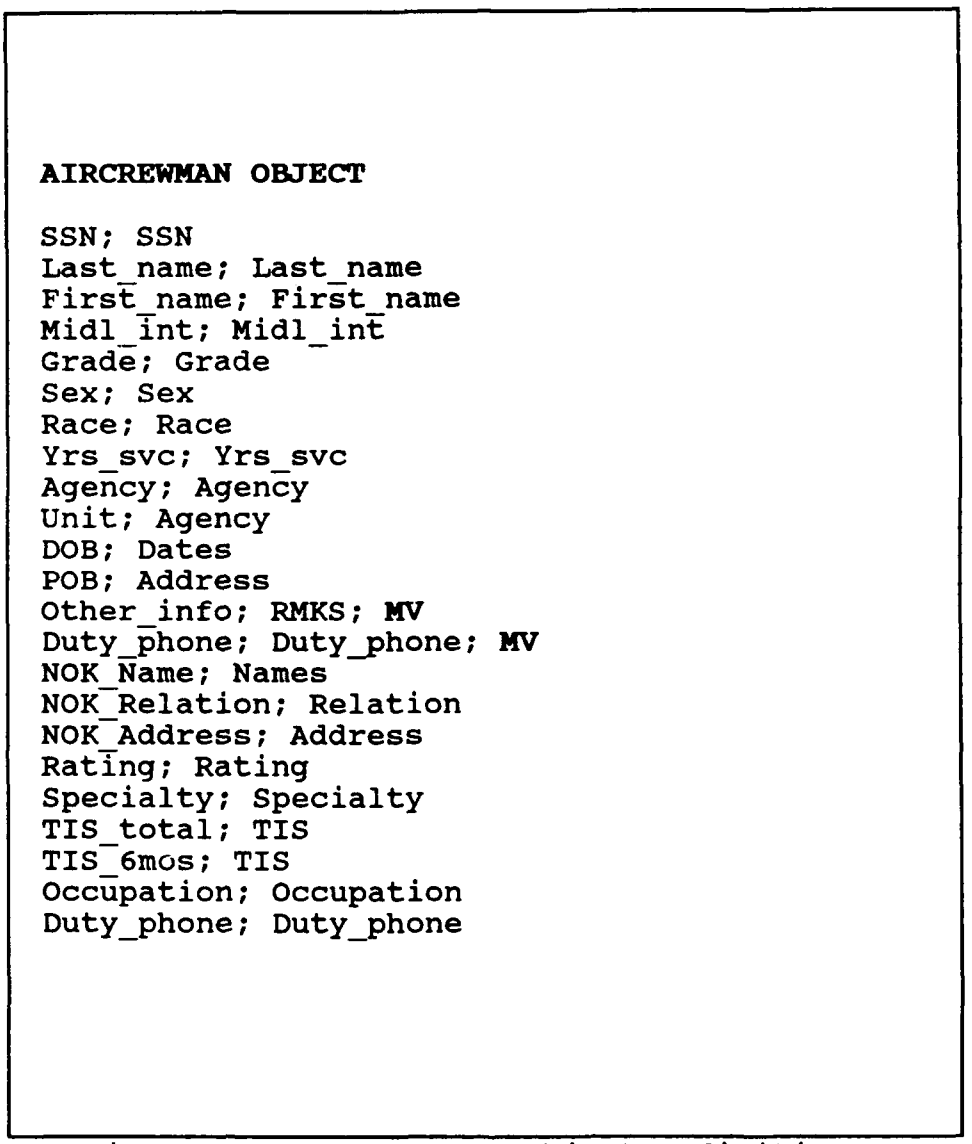


Figure 2.6: Sample FPS Object Definitions

an SSIS. Specifically, a domain definition contains a physical and a semantic description. The physical description indicates the data type of the attribute (e.g., numeric versus character or date, etc.) and any length, range or format constraints that are imposed on that attribute (e.g., allowable values are between 01/01/40 and 01/01/99 and must be displayed in the format given). The semantic description details the purpose or function of the particular attribute in order to distinguish it from other attributes (definitions) which may have the same physical description. For example, 01/01/50 could represent a date of birth, examination date, date forwarded, or some other attribute with a completely different meaning, such as number ordered/number received/number on hand.

b. **Object Definitions**

Object definitions are essentially an extension of object diagrams (see sample object diagrams in Figure 2.4). Each attribute of each object is listed with the domain from which its values may be drawn. If the attribute can be multivalued, this is also indicated. If an attribute is another object (an "object attribute" such as AIRCREWMAN in the MED_HIST and APPOINTMENT objects in Figure 2.4) the name of the attribute is capitalized. If only some of the attributes for an object attribute are to be included, the keyword "SUBSET" is used, with the set of attributes which are relevant listed in brackets following the capitalized name of the object attribute.

The discussion to this point has been without reference to any programming language and will, except for FPS examples, remain so. The products generated so far need not (and probably should not) be developed with an implementation language in mind. Requirements analysis and generation of a design to fit a particular language tends to unnecessarily constrain or bias the development process and will likely result in a less effective SSIS. Keeping an open mind throughout the process explores the greatest number of possibilities. This should yield the most robust design prior to selecting a programming language well suited to properly implementing it.

5. Requirements Review

Functional requirements should be substantially clear for an SSIS developed to this point. A brief digression for review is appropriate and recommended prior to proceeding with SSIS design. The developer should look over each product generated as well as the accumulated documentation. It is usually helpful to discuss progress and results with other interested individuals. Co-users/developers as well as people who will not be directly involved with the completed SSIS provide unbiased outside eyes to look objectively at what they see. This will frequently highlight errors or unseen opportunities that eluded the developer(s) because of intimate involvement with the process. The opinions of local software development professionals, if available quickly and inexpensively, are valuable as well.

The SSIS developer should also remain flexible and ready to make changes during these earlier phases; the further along a project is in the development process, the deeper the ramifications of changes will be and the harder they will be to make. Documentation should be routinely updated to reflect changes, adding remarks and notes explaining the reasoning that motivated them.

F. SSIS BASELINE (HIGH LEVEL) DESIGN

1. Schema (File) Design

"The structure of the entire database is called the *schema*, or the conceptual view." [Ref. 11]. The concept of schema, when applied to SSIS development, pertains to the conversion of objects into a computer-oriented format suited for managing the data that will be used in an application. Management of the data in an SSIS can be accomplished in a number of ways; through a database management system, variables, files, etc. There are several models for designing logical (conceptual) and physical data schema from objects identified during analysis. Among the more popular are the relational, heirarchical data, and entity-relationship models. In addition, there are various graphic tools (relation diagrams, object diagrams, relation tables, etc.) useful in visualizing objects and relationships between them during schema development. A detailed treatment of data modelling and its supporting tools is given in Reference 11.

At this point in development, the objects (data requirements) are fairly well focused. How data is managed in an SSIS (DBMS, files, variables, etc.) is largely a function of the programming language the developer selects once his design is complete. Further discussion of the details of schema/data and its implementation in an SSIS does not contribute appreciably to the following discussions and is left to the documentation which accompanies the developers' particular programming language.

2. Application Design

a. The Structure Chart (SC)

The structure chart (SC), as its name implies, is a design tool used to graphically represent the structure of an SSIS using modules, the relative organization of those modules and the information that passes into, between and out of them. Each module represents what will eventually be an identifiable segment of programmed code which performs one specific function or a limited set of functions within the SSIS. Figure 2.7 contains a portion of the upper level of the final FPS SC and serves to illustrate the following discussion. The FPS SC was built using Page-Jones' constructs as guidelines [Ref. 15].

The basic symbol of the structure chart is the square, which represents one module (function) in an SSIS. Modules are connected by lines which indicate relationships and direction of movement between modules. The modules are arranged hierarchically in both the vertical and horizontal directions. In general, higher level modules represent broad, executive functions which

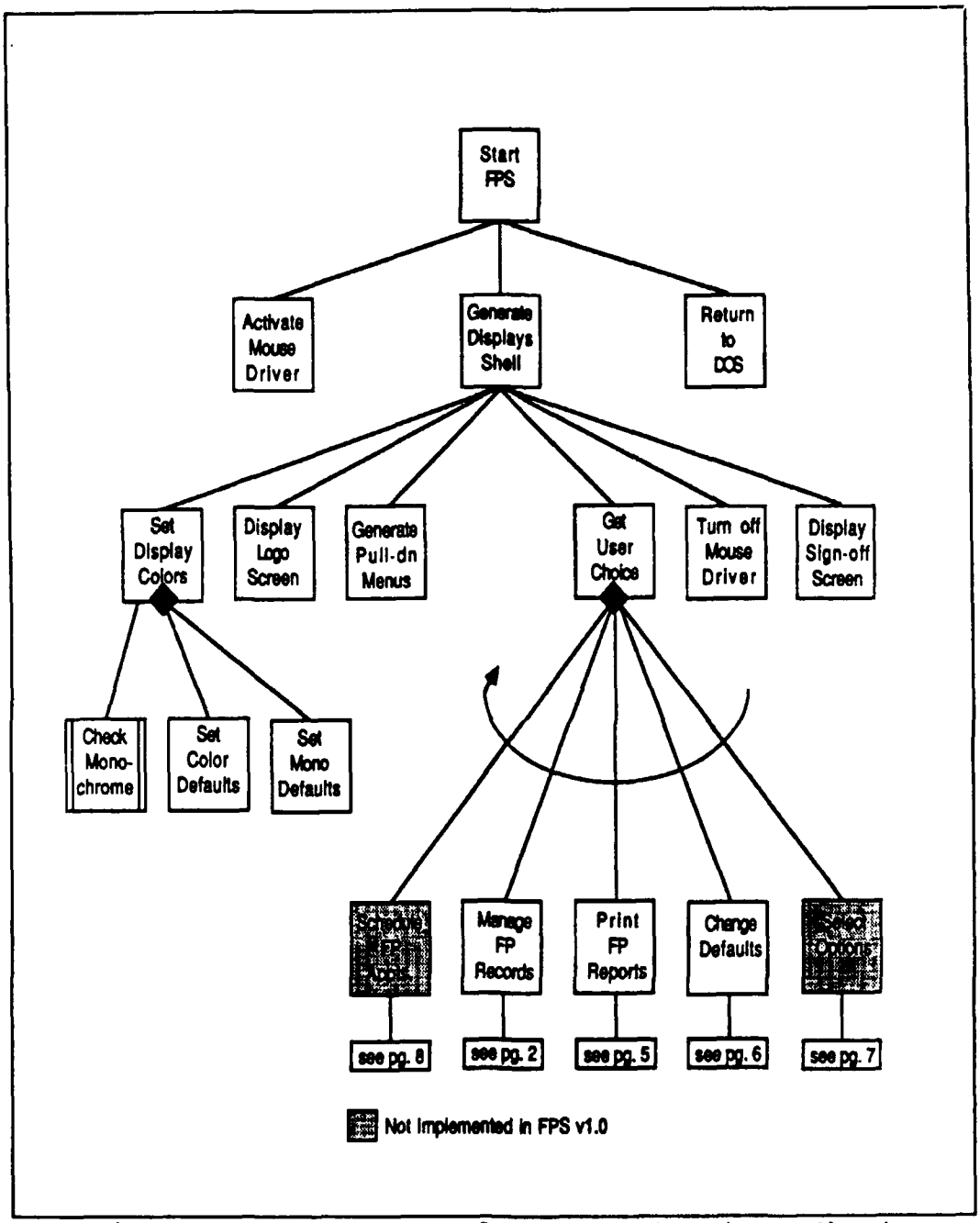


Figure 2.7: Upper Level of FPS Structure Chart

govern the increasingly detailed or specific functions discharged in subordinate level modules. In the horizontal direction, modules are organized to reflect the left-to-right order in which related modules (those connected by lines to common superior modules) execute their functions. This ordering generally reflects the sequence of operations that must occur in processing data through the SSIS in order to attain desired results.

It should be noted that the exact path traversed during execution of a completed application will vary with the options selected and data entered by the user. The SC does not show this variation in module execution or the flow of program control. It does however, show which functions are required once a certain task is directed by the user. For example, in FPS, once the user selects the "Generate Flight Physical Reports" option, the modules related to that option will execute in the order specified in the SC until another user decision is required (making the direction of program flow random again).

Given a basic familiarity with the SC and what it represents, one can be constructed for any SSIS project by breaking down the functions needed according to the guidelines given above for SC construction. Obviously, little detail (or accuracy, for that matter) can be expected in an initial SC. But, numerous revisions and additions quickly blossom from sparse beginnings as the developer's understanding of a system grows.

3. User Interface

The user interface is the method or methods an SSIS will employ to communicate and interact with its users. The user interface is also a primary means of regulating the degree of control users have in operating it. Through the design of input forms, menus and options, the degree of user control is regulated. It must not be overcomplicated or too restrictive or the user will become frustrated. On the other hand, it cannot allow too much freedom or the integrity of an application and its data may be violated (either inadvertently or intentionally). If it is too simplistic, users may be insulted or grow tired of repetitive remedial operations required to accomplish system tasks.

There are a variety of existing interface methods that can be employed singularly or in combinations. The interface(s) selected will depend on the tastes of an SSIS' users and the functions that must be performed. This is a complex area of considerable on-going research and many volumes of literature. Such complexity argues against having the casual developer attempt to develop any significant expertise in the area of man-machine interface. Instead, it appears more appropriate and productive for the casual SSIS user-developer to select appropriate interface style(s) from applications he is familiar with and imitate them in his design.

Complexity or sophistication in a user interface doesn't mean that it will be difficult to implement in an SSIS. Many complex interface softwares already exist and are easily adapted for inclusion in various applications. Many

programming languages are marketed with these problems already solved. User interfaces are often included as modules of code that are simply "spliced", as is, into developers' programs.

For FPS, a hierarchical system of pull-down menus is employed to allow the user to maneuver through the application. Once the desired general function (e.g., view/edit records, print records, etc.) is selected from the pull-down menus, different interfaces are employed to match the remaining detailed tasks to be accomplished (select particular records, confirm selections, etc.). This method was chosen for its simplicity and speed. Later, during implementation, it was convenient that the programming language's library of re-useable code contained several pull-down menu routines- already coded, tested and ready to use! Additionally, a module is included (from another library of re-useable modules of code) which allows FPS to use a two- or three-button mouse for cursor movement and activation of several frequently used keystrokes (ENTER, ESC and F1). If the user doesn't have a mouse device connected to his system or chooses not to use it, the keyboard's cursor movement keys can be used to the same end. The user may prefer the mouse for speedy manipulation of pull-down menus but be more comfortable using arrow keys while editing text at lower levels in the application. Multiple interfaces allow FPS to suit a variety of user tastes.

It is important to understand how critical the user interface is to the success of an SSIS. A well-conceived interface (or system of interfaces) will

strongly contribute to the success and longevity of an SSIS because it is friendly to use and users will likely use a friendly system. A poor interface (one that is difficult to use or understand or does not afford the user the freedom to control the application to the degree he requires) is almost certain to turn otherwise competent software into "shelfware" in short order.

4. Pseudocoding

Pseudocoding is an intermediate step that can be taken to bridge the gap between SC and the coding of SC modules using the selected programming language. As a rule of thumb, when the modules are translated into plain english statements of logic ("pseudocode") prior to actual programming, they should not be longer than one page. If a module's pseudocode is much longer than one page, it is likely that there is more functionality there than belongs in one module. Look for these opportunities in your SC, and break large modules into several smaller ones.

Pseudocode is a set of plain english statements which outline the logic and processes which occur in a module. It may also detail or imply the names of variables and parameters passed into, through and out of the module as well as its calling syntax. It makes no reference to a particular programming language, but may include statements in the form of common structured programming constructs (DO, WHILE, and FOR loops, IF clauses, etc.). Writing pseudocode for each module in the SC is recommended prior to actually coding them in the selected programming language. This is a good way to verify that

there are not too many functions packed into any single module. As each module's pseudocode is completed, file it with the documentation. Later, when programming is completed, the finished code replaces pseudocode in the documentation.

5. Programming Language/Development Tools

a. Selecting a Programming Language

There are a number of considerations that weigh into the selection of a language for SSIS implementation. Many of them stem from the particular functions to be delivered in any given SSIS. In general, the items following should be considered as part of the programming language selection process.

(1) Qualities Desirable in a Programming Language. The language should be able to support modularity and information hiding. The benefits of these constructs are discussed in detail by Page-Jones [Ref. 15] and others, and are generally considered essential to good applications development. The language must also be well-suited to the SSIS to be developed. If the SSIS is a database system (as was FPS), a database language is is probably best. If the application is more concerned with calculations or with the generation of detailed reports, consideration should be given to the scientific languages or languages with strong graphics and display capability, respectively. The language should

support the implementation of the SSIS design; it should not become a set of constraints that the design must be molded to fit.

Popular languages (dBASE, C, ADA, CLIPPER, COBOL, PASCAL, to name a few) may or may not be suited to a particular SSIS' implementation needs. There are advantages (provided the language fits the design) to selecting a tried and tested popular language. These languages tend to be well supported by their developers for updates, fixes and assistance with using them. There is also a good chance that others have written "how-to" books and tutorials for these languages that may be helpful in learning or becoming expert in their use. Popular languages are likely to enjoy greater longevity, helping to ensure the future viability and maintainability of applications developed using them. Finally, there is a greater likelihood that there are libraries of re-useable code generated for a popular language, which brings clear productivity benefits to the SSIS developer.

(2) Developer programming skills. Some programming languages are easier to learn and use than others. The developer who is experienced as a structured programmer has a wider range of options to choose from than does the new SSIS developer. The first time or novice developer should objectively consider his skills in the language selection process. He is probably better off to begin with a user friendly third/fourth generation language (BASIC, PASCAL, dBASE III/IV, etc.) than one of greater complexity (ADA, ASSEMBLY language, C, etc.). The simpler language offers the new SSIS

developer greater opportunity for success in implementation (i.e., code that he can make function to reflect his design intentions). There may be some cost or trade-off incurred in terms of quality and/or sophistication in the finished SSIS product. The simpler languages will likely have fewer and less refined capabilities than the more complicated ones, limiting SSIS development in some respects. But, once he becomes confident and experienced at this level, the developer can graduate in the development of future SSIS to the more complex and powerful programming languages.

(3) Languages in the organizational inventory. It is certainly cost-effective, all around, to use a language that is already in the organization's library. This is especially true if the developer is already familiar with programming in that particular language. There are also benefits to re-use of programming languages; re-use promotes the building of libraries of re-useable code, allows the potential to link applications within the organization, saves money in procurement of development software and improves productivity as the organization's developers become expert in the languages in the corporate library.

On the other hand, these benefits do not justify selection of an unsatisfactory programming language simply because it is available in the organizational library. If the languages available are not well-suited to the SSIS developer's requirements and design, new languages should be considered.

(4) Committing to a Programming language. Once a language is selected, it should not be considered a final and forever decision. For example, in the development of FPS, the project was initially implemented using dBASE III+. Later, unforeseen problems developed due to some language limitations and it became necessary to change to a different, less constrained programming language. There was a time penalty paid in learning and converting to the new language, but the ultimate success of the application justified the additional effort.

It is also possible that more than one programming language may be used in developing SSIS. The nature of an SSIS may demand the strengths of two or more languages to achieve all of the required functionality (e.g., graphics, complex reports, database functions, etc. in the same application). This poses some integration problems for the developer to solve in his detailed design, but the improvement that can be realized in the finished application are often worth the extra effort. In FPS, routines from the ASSEMBLY, C, CLIPPER, and dBASE III+ programming languages were integrated in the delivered prototype.

As a specific example, one ASSEMBLY language routine provided all the functionality for FPS to operate with a two- or three-button mouse. ASSEMBLY language was chosen for its ability to accomplish the required functions with a minimum amount of code. At the time, FPS was growing rapidly in size, and compact code became mandatory in the design if the

mouse-driven functionality was to be included. It is worthy of note that the actual "mouse" module was discovered in a library of re-useable code and that CLIPPER (FPS' main programming language) has the capability for integrating ASSEMBLY language code. The former demonstrates one of the benefits of modular design while the latter highlights one of the reasons CLIPPER was chosen as the main programming language for FPS.

(5) Learn the programming language. After the developer has selected his implementation language(s), he should spend the time to learn, in some detail, the programming language he has selected. In particular, learning the functions available in re-useable code libraries can significantly improve productivity when actually building the application. It is often tempting to skim the documentation and head impatiently into programming with inadequate knowledge. It is granted that gaining experience with the language is the best way to learn it. However, *it is not considered prudent to gamble the success of* an SSIS project by using it as the vehicle for learning a programming language.

b. Development Tools

For SSIS development, the kit of development tools is not elaborate. The toolkit employed in the development of FPS is summarized in Chapter I (see Figure 1.1). Minimum hardware requirements are the basic computer and a printer. Hardware beyond this is a largely a matter of developer tastes and budget, and generally encompasses convenience options not critical to developing SSIS to completion (mouse, additional storage devices, etc.).

Minimum software requirements will vary with the particular SSIS and the programming language chosen. Most programming languages are sold complete with all software required for programming applications, start to finish. Others are sold piecemeal or offer limited functions that rely on other commonly used software before they can be used for programming.

Given a programmer's text editor (a sort of word processor tailored to the programmer's needs) and his programming language software, the SSIS developer generally has all that he needs to implement an SSIS design. If the selected language dictates, separate linkers, compilers or optimizers may be required in order to complete programming. Other software such as automated code documenters, full-function word processors, etc. may be used at the developer's discretion to improve the appearance of his documentation, but do not generally contribute (beyond cosmetics) to effective design or implementation.

For FPS, the CLIPPER programming language was selected. CLIPPER came complete with libraries, compiler, linker, and various utilities software for programming convenience. The package did not include a programmer's text editor, which is required for generating source code. An additional linker was later added to the FPS tool kit (a faster linker was required during the development and testing of individual modules). Word Perfect 5.0 with it many capabilities proved to be well-suited for use in the generation of most of the FPS documentation.

6. The Computer Development Environment

Once the developer has decided on what will be in his toolkit of hardware and software, it must be assembled and configured at his workstation. The programming language software must be installed in accordance with the manufacturer's instructions. The text editor and any other supporting software must also be installed. Installation of software requires advanced planning to ensure that directories are properly partitioned and software is properly located to allow convenient and full interaction between software components. Time spent in carefully planning software set-ups will normally pay dividends later in programming speed and convenience for the developer. Additionally, it may be desireable to make batch files for frequently executed functions such as loading programming software, compiling or linking applications and any other repetitive operation conducted routinely in the course of programming.

G. SSIS IMPLEMENTATION

1. Top Down Implementation

The development of FPS from this point on was analogous to trimming a Christmas tree. Once a basic menu framework was established (the tree and its branches) ornamentation was added piece by piece (successively lower level modules from the structure chart). The development team worked through the SC in "top-down" fashion (from roots up, using the Christmas tree analogy), designing, building, testing, refining and integrating each module in the

order in which it was encountered in traversing the SC tree. It was more intuitive to deduce or evolve FPS by working from the general to the specific rather than trying to force the details before knowing the workings of the general executive functions. This introduces the next step, prototyping, which is the hub of the detailed design and implementation phase of the methodology.

2. Prototyping

The prototyping phase is the most time-intensive phase of the methodology, accounting for up to 50% or more of the time a developer spends on the SSIS development life cycle. Ironically, its description is short in comparison to those of previous phases. Prototyping is conveniently described as an iterative looping process which refines itself with each iteration until the final SSIS product is derived. Its strength lies in the rapid development of working modules of code (in place of the paper models of the traditional methodologies) which allow the user-developer to see actual working products and functions as they are created. With such rapid feedback, changes, flaws and errors become apparent for correction much sooner in the development life cycle and are corrected at a much lower cost. This, in turn, leads to earlier delivery of a more accurate system that will better satisfy its requirements.

The prototyping process is time-consuming primarily because of the testing/debugging that each module of programmed code undergoes as it is developed. This test/debug process is later re-applied to groups of related

modules as they are integrated and, ultimately, to the entire SSIS application when all modules are integrated.

The developer enters the prototyping loop by selecting one of the top modules from the SC. Next, the module is designed in detail based on the function(s) it must deliver, including the variables and data elements that the module will manipulate. This detailed design is in the form of pseudocode (discussed earlier) which clearly describes the logic and processes occurring in the module. Once a module is designed to the developer's satisfaction, it is coded using the programming language selected.

To test the coded module, a "driver" can be built which will simulate the conditions that would call on that module to perform its function. The driver is a block of code that will summon the module in testing, pass to it its required data and receive its output. It is helpful to code the driver in a manner which displays key values in, through, and out of the module so that errors (bugs) can be isolated. Testing continues until the module performs as designed. The driver is stripped off and the tested module is integrated with other tested modules. In similar fashion, drivers can be built to test and debug groups of related modules during multi-module integration.

The process need not necessarily proceed in top-down fashion. It may be that certain portions of the design will be implemented out of sequence in keeping with a project constraint or requirement. This process, coupled with

modular design constructs, provides the flexibility to accommodate these sorts of constraints.

The design-code-test-integrate loop continues until all modules in the SC have been successfully integrated. If severe problems are encountered in any iteration, the loop can be abandoned to begin fresh with a revised module design. If problems are discovered which drive a change in the high level design, the developer should take time out to update his documentation.

H. SUMMARY

Figure 2.8 summarizes the discussion contained in this chapter. It highlights the key steps and substeps of the foregoing methodology for economic development of SSIS. The following and final chapter discusses some of the lessons learned by the FPS development team in employing the methodology as well as the managerial concerns associated with using the methodology in an organization.

- **PREPARE FOR DEVELOPMENT**

- **FEASIBILITY STUDY**

- **REQUIREMENTS ANALYSIS**
  - Documentation
  - Requirements Identification
  - System Diagramming
  - Object/Data Requirements Identification
  - Requirements Review

- **BASELINE (HIGH LEVEL) DESIGN**
  - User Interface Selection
  - Pseudocoding
  - Programming Language Selection
  - Set Up Computer Development Environment

- **IMPLEMENTATION**
  - Module Prototyping
    - Design
    - Code
    - Test
    - Integrate

Figure 2.8: Summary of Development Methodology Steps

III. ISSUES, RECOMMENDATIONS AND CONCLUSIONS

A. SSIS DEVELOPMENT MANAGEMENT ISSUES

Shipboard managers include the Commanding Officer, Executive Officer, Department Heads, Division Officers and Enlisted Division Supervisors. Chapter I briefly described management's role in end-user applications development. The issues of organizing, planning, leading and controlling will now be discussed in more detail. These issues will be addressed from the viewpoint of management in the shipboard environment.

1. Organizing

The first step in the organizing effort is to see what resources are available. Many of the hardware/software resources are already in place onboard the ship or readily available; however, the most important resource, trained SSIS developers, must in many cases be "home-grown". Questions such as the following must be asked:

- Are there personnel onboard with undergraduate or graduate training in Information Systems or Computer Science?
- Are there personnel onboard who have developed computer-based SSIS applications?
- What percentage of the crew have experience using a microcomputer?

Personnel identified through the answers to these questions will form the foundation for a SSIS development program. As mentioned earlier, personnel are the most important resource and these types of individuals will prove critical for success in this new endeavor.

Once this foundation of "computer literate" personnel have been identified, they in turn must be used to improve the computer literacy of other crew members. After all it is these crew members who will share the applications that you develop. Studies by Rivard and Huff [Ref. 16] have proven this and they conclude "...the fact that users with better computer background have a more positive attitude toward user development applications suggests that attempts should be made to improve general computer literacy of users prior to their . . . undertaking development activities."

Finally, if a highly trained programmer or systems analyst is not readily available (onboard ship), it will be helpful to establish a rapport with one at the nearest NARDAC (for example). Having one of these professionals just "a phone call away" will assuredly add to the success of the SSIS development program.

2. Planning

Shipboard managers must decide whether an SSIS meets specific needs and whether or not it will benefit them in achieving organizational (shipboard) goals. A feasibility study should be completed (see Chapter II) to help make these decisions. These managers must plan for the resources required

for SSIS development. Figure 3.1 suggests a list of items the manager should consider to plan for SSIS development in the shipboard environment. Figure 3.2 lists the actual costs incurred for developing the FPS project. These costs are typical to those expected to occur in developing an SSIS aboard ship. Once the resources listed in Figure 3.1 are on hand, the next important planning item that must be considered before building an SSIS is the development of proper guidelines. Such guidelines, according to Alexander [Ref. 2], include efforts to ensure the developer tests and documents his application(s) in accordance with standards managers establish. If managers fail to implement these guidelines, poorly developed applications will likely evolve and unnecessary duplication of effort will probably take place. For example, NARDAC Norfolk estimates [Ref. 9] that there are 75,000 to 100,000 applications developed by end-users in the Navy and that at least 80 percent of them are duplicates of other efforts.

To help reduce this duplication of effort, NARDAC Norfolk has developed a program called Multi-Site Customized Application Software (MS-CAPS). Software such as Armory Inventory Control System, Personnel Management System and Technical Publication Library Tracking plus many more are available free of charge via the MS-CAPS program. Applications developed by end-users in the fleet should be sent to NARDAC for inclusion in MS-CAPS, thus becoming available to other commands. NARDAC's Remote Bulletin Board System is another method to reduce duplication of effort by offering various applications and utilities written in dBase, Clipper, Pascal and C. This system

	ITEM	COST
Hardware:	Microcomputer w/monitor + hard drive	$ 2500
	Printer (dot matrix or laser)	$ 300-1000
	Mouse (optional)	$ 100
Software:	Operating System (ex: MS-DOS 3.3)	$ 115
	Text Editor (ex: Word Perfect, QEdit)	$ 100-600
	Development Tool/Programing Lang. (ex: DBase IV, Clipper, Paradox, FoxBase, Magic PC, Oracle, Ingres)	$ 200-900
	Compilers and Linkers (ex: Clipper, dBFast, MicroSoft or Borland Linkers)	$ 75-600
Support Materials:		
	Computer Disks	$ 100
	Printer Paper	$ 100
Training Examples:		
	DBase IV Tutorial	$ 70
	Using WordPerfect (VHS Cassette)	$ 25
	Reference Guides**	$ 150
	NARDAC Courses:	
	Intro to Programming	$ 395
	Programming with dBASE	$ 375
	"C" Language Programming	$ 450
Suggested Periodicals:		$100-200/yr
	Data Based Advisor	
	InfoWorld	
	Software Digest	
	CHIPS	

Figure 3.1: Resources for SSIS Development

ITEM		COST
Hardware:	Microcomputer w/monitor + hard drive	$ 0*
	Printer (dot matrix)	$ 0*
	3-button Mouse	$ 75
Software:	Operating System (MS-DOS 3.3)	$ 0*
	Text Editor (Word Perfect and QEdit)	$ 0*
	Development Tool/Programing Lang. (Clipper)	$ 160**
	Compilers and Linkers (Clipper and MicroSoft Linkers)	$ 0*
	Art Fuller's Clipper Library	$ 20
Support Materials:		
	Computer Disks	$ 70
	Printer Paper	$ 75
Reference Guides:		
	Dynamics of Clipper	$ 22
	Programming in Clipper	$ 33
	Mastering Word Perfect 5.0	$ 21
	DBase Programmers Ref. Guide	$ 24
	TOTAL:	$ 650

Notes:

* Developers owned items marked with single asterisk prior to FPS project.

** Clipper purchased @ $ 80 per copy (student discount rate) normal price $ 695.00

Figure 3.2: FPS Development Costs

permits downloading of the source code that can be modified for use in the SSIS under development.

Managers should also establish a plan and guidelines for building and updating onboard libraries of re-useable code modules. The system devised should be simple to use, easy to access and be governed by a set of standard formats that ensure maximum opportunity for re-use of all modules.

3. Leading

One of the most important and sometimes least available commodity aboard ship is time. In light of that statement, the last thing a shipboard manager wants to hear is that he has another job (SSIS development). SSIS development must not be viewed from such a narrow perspective. The truth is, automation of any process will likely save time in the long run.

Rivard and Huff [Ref. 16] assert that in order for end-user SSIS development to be successful, the developers must be supported by good managers. Another type of support that leads to greater user satisfaction are user support groups. Informal groups such as these should be initiated onboard ship to establish contacts and share ideas between experienced and novice users/developers.

Realistically speaking, in today's Navy a professional SSIS development officer designator or enlisted rating does not exist. Data Systems Technicians onboard ship repair and maintain sophisticated computer equipment, but developing software is not part of their job description. What this means is

SSIS development efforts will be accomplished by any officer or enlisted personnel trained to do so or who is willing to give it a try. In leading these efforts, managers must ensure personnel selected for these duties are given adequate time, incentive and an environment to successfully develop an SSIS.

Time for these development efforts is contingent upon a schedule that will allow periods away from the individuals primary duties to be spent on the SSIS. Incentives may include command recognition or awards from the Navy's Beneficial Suggestion Program for the savings in man-hours that SSIS developers have brought the Navy. Finally, a creative work atmosphere or environment must be established. A space onboard the ship should be devoted to organize a workstation to include the necessary hardware, software and reference materials for successful SSIS development.

4. Controlling

"Control" is a buzz word used when computer professionals discuss end-user applications development. Specifically, it pertains to managers controlling the application development efforts of user-developers. There are four major areas of control shipboard managers must consider when undertaking an SSIS development, these areas are: tracking/reporting, periodic reviews, productivity and quality.

a. Tracking/Reporting

Tracking the process of the SSIS development is extremely useful for both the immediate project supervisor as well as the entire chain of command. Tracking is also valuable for individual developmental efforts as the information will be applied for future projects. There are several planning/control tools available for tracking and reporting projects.

PERT charts were developed in the 1950's to plan and control weapons development projects for the Navy and are still useful today. The PERT chart is a graphical networking tool which gives managers a visual representation of the various events taking place during project development. Furthermore, PERT charts display the times to complete each event and graphically display the order of completion of the events.

Another useful tracking and reporting tool is the Gantt chart. The Gantt chart is a simple time-charting graphic tool. The SSIS development process can be easily scheduled and its progress evaluated using this tool. By analyzing the Gantt chart with the current date in mind, one can get a "snapshot" of the entire project's status (which phases are ahead/behind schedule). A Gantt chart for the FPS project is provided in Figure 3.3.

PERT and Gantt charts have their advantages and disadvantages. For the most optimum tracking and reporting efforts, Whitten, Bentley and Ho [Ref. 20] "...recommend that PERT and Gantt charts be used in a complementary manner to plan, schedule, evaluate and control systems

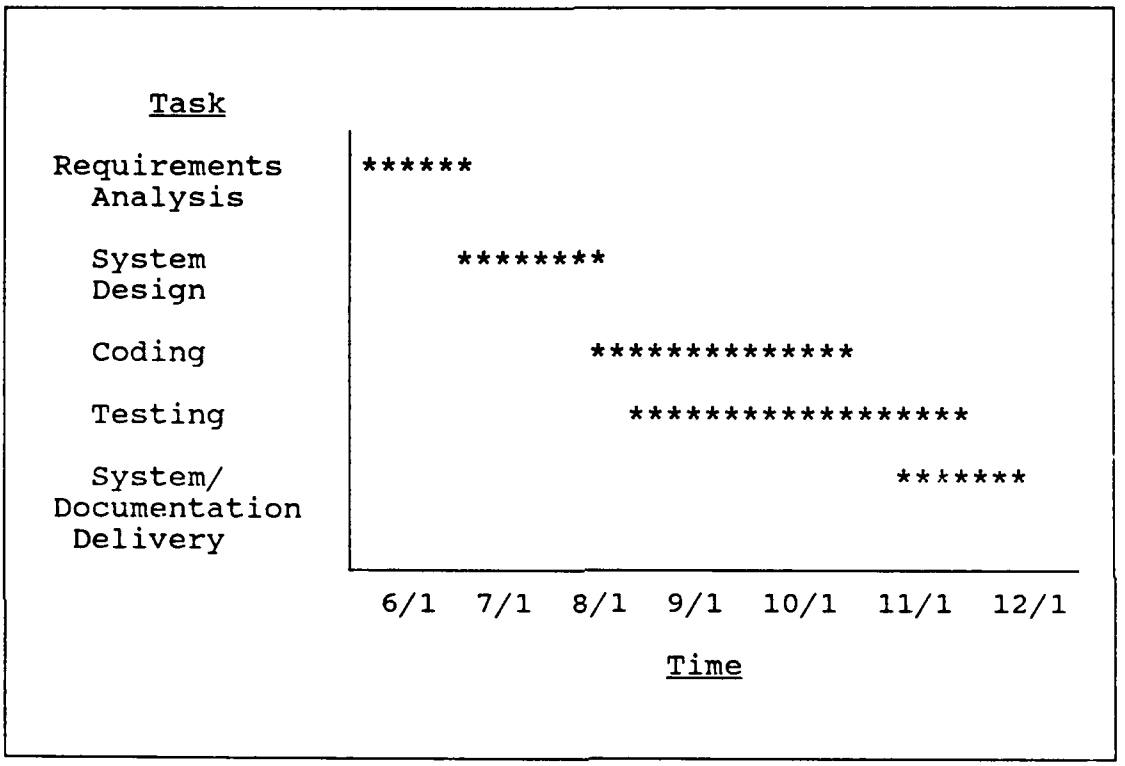


Figure 3.3: FPS Gantt Chart

development projects." Detailed information on PERT and Gantt charts can be found in any textbook on the subject of management.

b. Periodic Reviews

The feasibility of continuing or abandoning the project must be frequently monitored. This monitoring process is called a periodic review or feasibility analysis. At the completion of this review, the manager can revise the estimated phase completion dates (see Figure 3.3), apply more or remove personnel from the project, or abandon the project altogether. Whitten, Bentley and Ho [Ref. 20] have identified five feasibility checkpoints in which reviews must take place. They are:

- Survey Phase Checkpoint: Does the problem identified arrant spending dollars for an SSIS development effort?

- Study Phase Checkpoint: After cost/benefit analysis, is the problem identified still worth solving with an SSIS? If so, allocate funding and continue with the project.

- Evaluation Phase Checkpoint: Identify alternative solutions. For example, modify existing system, purchase SSIS package or design and construct a custom SSIS.

- Selection Phase Checkpoint: Select the hardware and associated software if required. Today, in most cases, the hardware will be in place aboard ship. If this is not the case, consult the Supply Officer for hardware purchasing decisions.

- Design Phase Checkpoint: Upon completion of the design phase, the complexity of the SSIS is well understood. Coding and testing are the next

phases and this design phase check-point allows a final chance to update estimated completion dates and reevaluate feasibility.

Note that the project is not left without management involvement on completion of the design phase checkpoint. Tracking and reporting project status will continue until its completion.

c. Productivity

Another important management control practice in software development is measuring and monitoring the productivity of the developers. After all, the more productive the developer, the less expensive the software product he will produce. In order to effectively measure productivity, we must first define it. Boehm [Ref. 3] defines productivity as the outputs produced by the process divided by the inputs the process consumes. Therefore productivity can be improved by either increasing this output or decreasing the inputs. Figure 3.4 summarizes Boehm's [Ref. 3] definition of the inputs and outputs of the software development process.

For many years, lines of code has been the standard for measuring the output of the software development process and has received some criticism. Boehm [Ref. 3] lists six deficiencies to using lines of code as a software metric (output measuring device); however, it is an adequate output measurement for an SSIS development because of its relative ease to define and measure. Once productivity measurements (both input and output) have been established,

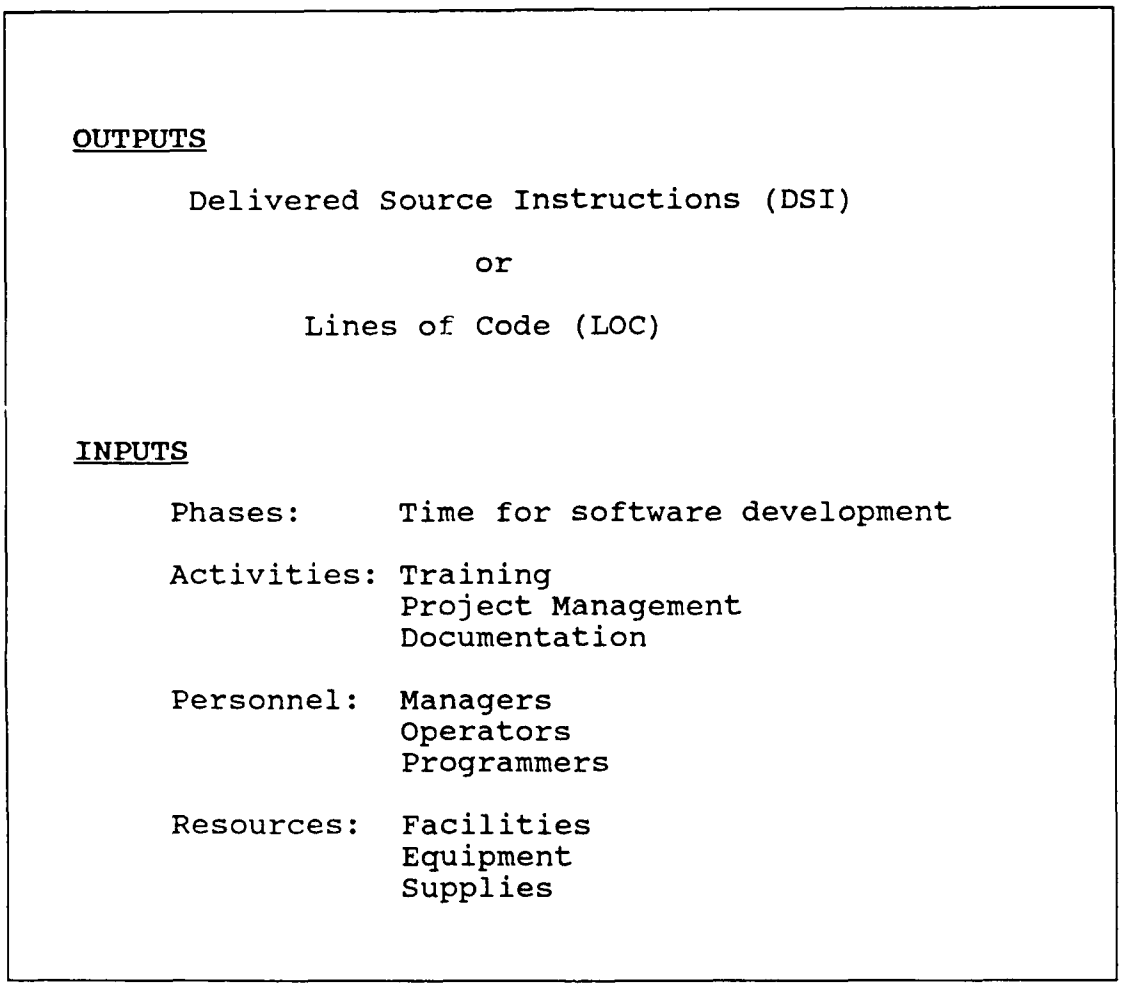


Figure 3.4: Productivity Inputs and Outputs

the manager must then strive to improve productivity. Boehm [Ref. 3] has indicated several methods to improve productivity.

The first and foremost method is to use the best personnel (highly trained and motivated) to develop the SSIS. Second, eliminate unnecessary steps in programs and avoid rework by reusing components whenever possible. Third, make the steps performed by the code more efficient by using the best software tools. Finally, build simpler products and use techniques such as rapid prototyping, as discussed in Chapter II.

d. Quality

O'Leary [Ref. 14] conducted a survey of 475 information center managers from Fortune 1000 Aerospace and Government organizations and observed that these managers are concerned over the lack of control and training of end-users in the application development process. One lesson to be taken from O'Leary's study is that shipboard managers ought to be prepared to control SSIS development efforts as a measure for ensuring quality in developed applications.

During the development process, managers should take time to assess quality of the SSIS by answering the questions suggested below. If the answer to any of these questions is yes (indicating that an unreasonable effort would be required to ensure satisfaction of the given quality attribute), then there may be doubt about the quality of the SSIS. These quality factors (questions) are

taken directly from McCall's [Ref. 13] work in 1977 and are still relevant to software development today. These questions are:

- Maintainability: Will it be difficult to correct errors found in the software?
- Flexibility: Will it be difficult to change the software?
- Testability: Will it be difficult to test the software?
- Reusability: Is it difficult to re-use components?
- Correctness: Does the software fail to do what it's supposed to do?
- Reliability: Does the software fail to perform accurately all of the time?
- Useability: Is it hard to run the software?

Controlling quality during the software development process is a continuing effort. Managers must ask the types of questions listed above during the periodic review process. It's the developer's duty to have these quality factors in mind when constructing each module. If all personnel concerned in the development effort stress quality, a successful SSIS will more likely result.

B. FPS LEVEL OF EFFORT SUMMARY

At this point, the reader may question what type of effort in man-hours goes into the development of a typical SSIS. The Flight Physical System (FPS) was developed (as discussed in Chapter II) in order to examine if SSIS development without professional programers or analysts is feasible and to propose a methodology for doing so. The FPS project succeeded on both counts.

Figure 3.5 summarizes the level of effort involved to construct and implement FPS.The effort in man-hours was collected from the notes, diaries and logs of the two developers.

When reviewing the data in the Figure 3.5 keep the following facts in mind. For a shipboard developed SSIS, little or no hours will be spent on requirements analysis because you will be developing the system for yourselves (end-user) and you know your requirements. Many of the FPS requirements analysis hours were spent with the Flight Surgeons (end-users) deciding what they wanted (their requirements) in this system and documenting these requirements. An overall timeline for the case study project was presented earlier in the FPS Gantt chart (Figure 3.3). As in the shipboard SSIS development scenario, the SSIS development was not the primary duty of the developers, explaining the relatively long timeline in the Gantt chart compared to the total man-hours of development effort.

C. FPS DEVELOPMENT PROBLEMS AND LESSONS LEARNED

The first and foremost problem faced in the development of FPS was learning the programming language selected. The developers had limited classroom and practical experience in structured programming methods as stated in Chapter I; however, the developers had virtually no background in the orginial programming language selected for FPS (dBASE III Plus). Reading the various

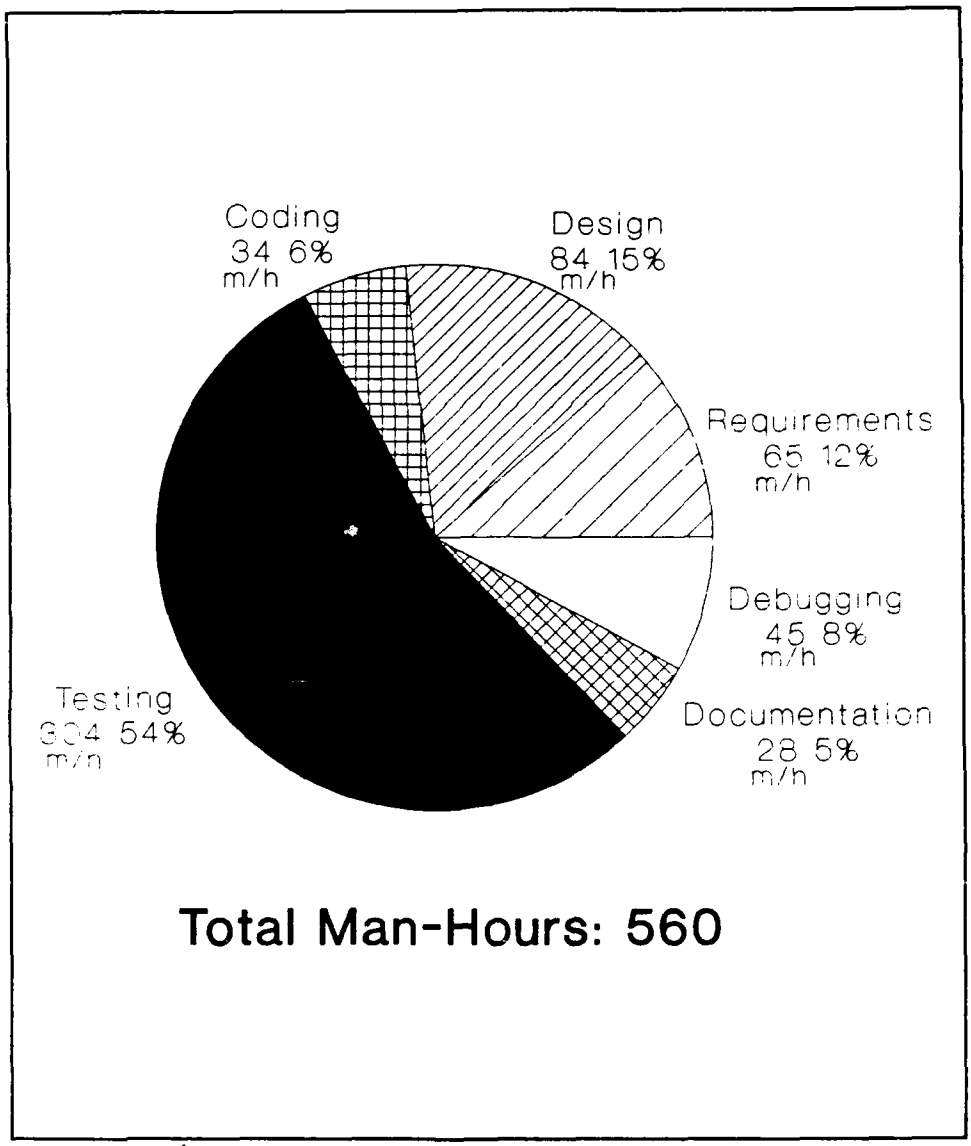


Figure 3.5: FPS Development Effort

reference guides (see Figure 3.2) available helped overcome this "learning curve" and the structured programming classroom background made the material much more understandable. The bottom line is, with no experience, the learning curve is steep but achievable and with a background in one programming language, the transition to another is relatively smooth.

The next problem faced by the FPS developers was a decision to change to another (more powerful) programming language in midstream. Using dBASE III+, a functionally correct but lackluster prototype was developed. Unfortunately, this prototype was slow to execute instructions and access data files, making uncomfortably long delays for the user. It was considered weak in user-friendliness in its interface, its help facility was inadequate to guide the user through all phases of program operation and, due to a dBASE III+ limitation, was unable to support the number of data fields required for each record (a fatal flaw). The users and the developers agreed that improvements were required and that a shift to an alternative programming language/database system was necessary to support the FPS data requirements. In order to gain these improvements, the FPS developers changed to the Clipper programming language (see Chapter II for details). Changing programming languages midway through a development effort may delay the completion date as the learning curve problem appears again but this decision must be made if the designed SSIS product cannot be constructed with the original language. The bottom line again is, if a change in programing languages is required to achieve the desired SSIS, then do it.

A final problem faced by the FPS developers was the interaction with the end-users of the SSIS (Flight Surgeons). The attitude or involvement of the these end-users can best be described as hot and cold. During the requirements analysis phase of the project the end-users were very cooperative and excited about their future SSIS. They were eager to provide input and information required by the developers. As time went on, this excitement diminished as normal work priorities arose. When the first FPS prototype was demonstrated, the end-users again became interested pointing out features they liked and disliked about the SSIS. The bottom line for dealing with this problem is don't expect too much from the users of the SSIS. These end-users have jobs also and frankly don't spend a whole lot of time thinking about the SSIS unless the developer is there with them. Of course the preceding paragraph does not apply when the developer is the end-user of the SSIS.

Lessons learned in the FPS development effort are simple and can be summarized as follows: Keep thorough records on everything you do. Take notes or keep a diary of all action items, decisions, ideas, costs, and just general daily remarks on the SSIS effort to date. Tape recording thoughts is another method of annotating items of use for a later date. The commitment to this type of a documentation effort proved invaluable to the FPS project. FPS developers were constantly referring to these notes and using the data for the tracking and reporting issues discussed earlier in this chapter.

D. CONCLUSIONS AND RECOMMENDATIONS

The techniques presented in this thesis are beneficial to the U.S. Navy, other military services as well as any organization in general for establishing a SSIS development setting. These benefits include background information on the history of applications development, a detailed walk through of an SSIS development process from identification of the need of the SSIS to delivery of the final product. Note this walk through or demonstration was accomplished without the aid of professional programmers or analysts, a primary goal of the research. Other benefits include management concerns such as resources and cost associated with SSIS development as well as planning, leading, organizing and control issues.

It was the intent of the research to present a methodology that allows timely and cost-effective solutions to small-scale management information problems. As the term methodology indicates the techniques presented are just "methods". Interested SSIS developers are encouraged to use what they like and discard what they don't like about the methodology in order to build the most effective SSIS for their organization.

The gap between the traditional applications development methodologies of the past and the 4 and 5GL's of the 1990's is narrowing rapidly. This gap will be bridged as computer literacy and end-user computing continue to rise and more and more non-professional programmers begin developing SSIS's.

APPENDIX

6 June 1989

MEMORANDUM

From: LCDR John Ash and LT Dale Spaulding

To: Professor Barry Frew (Academic Associate)

Subj: Thesis Feasibility Study

Encl: (1) Thesis Feasibility Study

1. Thesis title: Flight Physical System (FPS)

2. General area of thesis effort: Software system development life cycle (SDLC) for a database system and associated applications programs necessary to support the administration, documentation and tracking of flight physicals conducted by US Army flight surgeons at Fort Ord.

3. Enclosure (1) is a study of the above thesis' feasibility.

4. This study is unclassified.

A. BACKGROUND

The three flight surgeons assigned to the Fort Ord Aviation Medical Clinic are responsible for maintaining records and conducting annual flight physicals for approximately 1500 military aircrewmen assigned within their area of responsibility. For each aircrewman's annual physical, there are two medical forms generated to document all results and the examining physician's recommendation regarding each aircrewman's medical eligibility for flight duty. These forms must be forwarded to the US Army Aeromedical Activity (AAMA) in Fort Rucker, Alabama where they are reviewed for completeness and accuracy by other flight surgeons who make the final determination of each aircrewman's medical eligibility for flight duty. After review, each set of forms is manually transcribed into a central computer database at the AAMA.

The system in force at the Fort Ord clinic for flight physicals administration is fully manual. Each set of forms for each aircrewman is re-generated and processed for each annual flight physical. The clinic's flight physical database is in the form of some 1500 individual medical records for the aircrewmen served. It is maintained by a small staff of civilian and Army enlisted personnel. A large part of the physicians' time is spent in processing and handling the medical exam forms (minor calculations, identifying exam results that are out of prescribed limits, re-generating mandatory standard comments on the forms, filling in standard entries, checking accuracy and completeness, etc.).

In spite of the significant time and effort dedicated to ensuring complete and accurate exam forms, more than 8000 forms are returned by the AAMA annually to the various clinics for correction and re-submission. This results in excessive postage costs, significant flight surgeon time wasted in administrative re-work (largely due to minor errors and omission of mandatory data in the forms) and inordinate delays in final determination of affected aircrewmens' eligibility for flight duty.

The flight surgeons at Fort Ord Aviation Medical Clinic need an improved system for the administration, documentation and tracking of flight physicals conducted. This is the problem that the proposed thesis is intended to address and solve.

B. PROBLEM STATEMENT

The proposed thesis must solve, (to the clients' and thesis committee's satisfaction), the problem addressed in the foregoing discussion not later than 15 March, 1990 using only the resources indicated below (see paragraph F.3).

C. PROJECT SCOPE

1. General Scope:

Through automation and a systematic analysis and design approach, we propose to develop a system which will significantly reduce the administrative overhead, error rate and cost of documenting and processing flight physical exam data at the Fort Ord Aviation Medical Clinic. Beyond this core goal, there are other benefits which can be derived from the system envisioned. These include the ability to automatically generate and maintain a tickler file of aircrewmen due for flight physicals (not feasible using the current system), an examination appointments scheduling system, the ability to forward aircrewmens' aeromedical histories by disk or modem, improved readability of typed machine-generated forms over forms hand scribed by physicians (not an insignificant benefit), the potential to eliminate hardcopy forms altogether by forwarding exam data to the AAMA electronically and the associated potential for reduction/elimination of AAMA data transcribers (a savings of $150,000/year for a 50% reduction).

2. Detailed Scope:

The project will progress through the phases of the following system development life cycle (SDLC), culminating in a fully functional and implemented system for the user:

I. ANALYSIS: (est. 30% of total effort)
- -requirements analysis
- -system prototyping
- -requirements document review

II. DESIGN: (est. 40% of total effort)
- -system design
- -design document review
- -hardware specification and procurement (user funded)

III. CODING and TESTING: (est. 20% of total effort)
- -coding of application program(s)
- -testing and revision

IV. IMPLEMENTATION: (est. 10% of total effort)
- -smooth owner's manual (deliver)
- -user training
- -hardware hook-up, test and start-up

-database start-up
-system implementation (parallel ops)
-on-line system testing
-system delivery

D. PROJECT CONSTRAINTS

1. Project must be completed no later than 15 March 1990.
2. Project team limited to ASH and SPAULDING.
3. Project team members have limited software system analysis and design experience.
4. Project team members have limited programming experience (PASCAL and BASIC only).
5. Project team members must satisfactorily complete curriculum course requirements and maintain an acceptable grade point average concurrent with thesis work.
6. The client needs the system as soon as it can be delivered.
7. Resources limited to those described in paragraph F.3.

E. ALTERNATIVE SOLUTIONS

1. Abandon the project due to infeasibility.
2. Limit project scope to requirements phase only; leaves client to contract for the remainder of system development elsewhere.
3. Limit project scope to requirements and design phases only; leaves client to contract for the remainder of system development elsewhere.
4. Re-define project scope to specify development of an improved non-automated system (minimizes the potential negative impact of project team member inexperience constraints).

F. COST/BENEFIT ANALYSIS

1. DEVELOPMENT COSTS:

(a) Administrative Supplies: $600.00

(computer paper, binders, floppy disks, printing, photocopying, etc.)
(Funded by project team)

(b) Travel Expenses: $100.00

(Gas expense for travel to and from Fort Ord)
(Funded by project team)

(c) Project Team salaries: $6768.96
(96 man-days x $70.51/day)
(Funded by taxpayers)

(d) System Hardware/Software $3400.00
(Funded by client)

TOTAL: $10868.96
funded by others: . . . . . - $10168.96
est. project team costs: . . . . . **$ 700.00**

(e) Project Team time estimates (in 8 hr. man-days):

Requirements phase (30%): 28
Design phase (40%): 38
Coding phase (20%): 19
Implementation phase (10%): 10

EST. TOTAL TIME REQUIREMENT (man-days): 95

2. BENEFITS:

(a) Project team members fulfill MS degree requirements.
(b) Thesis committee opportunity for publishable material.
(c) Client benefits:

-Annual time savings- manual form preparation: $1750.00
(aircrewman time)

Manual: 10min X 1500/yr = 15000
Automated: 3min X 1500/yr = 4500
Savings (minutes): = 10500/60 = 175hrs
Benefit: 175hrs X $10/hr = $1750/yr

-Annual time savings- manual form preparation: $2000.00
(flight surgeon time)

Manual: 10min X 1500/yr = 15000
Automated: 2min X 1500/yr = 3000
Savings (minutes): = 12000/60 = 200hrs
Benefit: 200hrs X $10/hr = $2000/yr

-Annual mailing cost savings: $75.00

150 returns/yr X .50(two way)= $75/yr

-Annual time savings in form rework: $250.00
(flight surgeon and staff time)

150 returns X 5min/form = 750min/60 = 12.5hrs
$ Benefit: 12.5hrs X $20/hr = $250/yr

TOTAL ESTIMATED ANNUAL SAVINGS $4075.00

3. RESOURCES AVAILABLE:

(a) PROJECT TEAM TIME:

8 thesis classroom hours/week	8
x 12 weeks/quarter	x12
	96
x 4 quarters remaining	x4
	384
2-out-of-class hours for every in-class hour:	+384

TOTAL PROJECT TEAM HOURS AVAILABLE: 768
8 hr. man-days: 96

(b) IS/CS department staff analysis, design and programming experience (as required; offsets constraints due to project team inexperience).

(c) PROJECT TEAM FUNDING: sufficient personal funds available to support project team costs specified above.

G. RECOMMENDATION:

Proceed with thesis project in accordance with paragraph C.

Justification:

(a) Project is **Operationally Feasible**: The problem is worth solving (full and successful MS degree completion is contingent upon satisfactory thesis completion). All project stakeholders (project team members, clients and

thesis committee) strongly support implementing the proposal in paragraph C as a feasible solution to the problem stated in paragraph B.

(b) Project is **Technically Feasible**: Adequate technical and technological resources are available to allow a practical solution to the problem. Time available is sufficient to ensure project completion (estimated time requirement equal to time available).

(c) Project is **Economically Feasible**: The expenses (time and money) that must be absorbed by the project team are considered small; since a thesis must be completed to earn a degree, and any thesis will result in comparable personal expense, these costs are considered sunk. Further, as seen in paragraph F, the benefits for all stakeholders significantly outweigh project costs.

LIST OF REFERENCES

1. Alavi, M., "An Assessment of the Prototyping Approach to Information Systems Development", Communications of the ACM, Vol. 27, No. 6, June 1984, pp. 556-563.

2. Alexander, M., "Program - Training = Problem: End-users' enthusiasm can lead to duplicated programing, glitches", Computerworld, Vol. 22, No. 42, October 17, 1988, pp. 43-44.

3. Boehm, B., "Improving Software Productivity", IEEE Computer Magazine, Vol. 20, No. 9, September 1987, pp. 43-57.

4. Burns, R., and Dennis, A., "Selecting the Appropriate Application Development Methodology", Data Base, Vol. 17, No. 1, Fall 1985, pp. 19-23.

5. Connor, M., Structured Analysis and Design Techniques, Softech, Inc., Waltham, Massachusetts, 1980.

6. DeMarco, T., Structured Analysis and System Specification, Yourdon Press, New York, New York, 1979.

7. Fuller, A., Dynamics of Clipper, Dow Jones-Irwin, Homewood, Illinois, 1989.

8. Gupta, R., Cheng, W., Gupta, R., Hardonag, I., and Breuer, M., "An Object-Oriented VLSI CAD Framework, A Case Study in Rapid Prototyping", Computer, Vol. 22, No. 5, May 1989, pp. 28-36.

9. Hamblen, D., "Meet the New OP-945: RADM Paul E. Tobin", CHIPS, April 1989, pp. 4-7.

10. Headquarters U.S. Army Aeromedical Center, Fort Rucker, Alabama, "Flight Surgeon Office Automation Program Memorandum", 20 October, 1988.

11. Kroenke, D. and Dolan, K., Database Processing: Fundamentals - Design - Implementation, Science Research Associates, Inc., Chicago, Illinois, 1988.

12. Martin, J., Application Development Without Programmers, Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1980.

13. McCall, J., Richards, P., Walters, G., "Factors in Software Quality", NTIS AD-A049-014, 015, 055, November 1977.

14. O'Leary, M., "IC's are losing development control, survey shows", PC Week, Vol. 5, No. 36, September 5, 1988, pp. 4-5.

15. Page-Jones, M., Practical Guide To Structured Systems Design, 2nd ed., Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1988

16. Rivard, S., Huff, S., "Factors of success for end-user computing", Communications of the ACM, Vol. 31, No. 5, May 1988, pp. 552-561.

17. Stoner, J. and Wankel, C., Management, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1986.

18. Tobin, P., Oral presentation to computer technology curriculums, Naval Postgraduate School, September 26, 1989.

19. Watterson, K., "Case Study: City Works", Data Based Advisor, Vol. 7, No. 9, September 1989, pp. 102-104.

20. Whitten, J., Bentley, L., Ho, T., Systems Analysis and Design Methods, Times/Mosby College Publishing, St. Louis, Missouri, 1986.

21. Yourdon, E. and Constantine, L., Structured Design, Yourdon Press, New York, New York, 1978.

INITIAL DISTRIBUTION LIST

1. Library, Code 0142 — 2
   Naval Postgraduate School
   Monterey, CA 93943-5002

2. Timothy J. Shimeall — 5
   Department of Computer Science
   Code 52SM
   Naval Postgraduate School
   Monterey, CA 93943-5002

3. LT Dale R. Spaulding — 2
   Navy Regional Data Automation Center
   Norfolk, VA 23511-6497

4. LCDR Robert L. Knight — 1
   Department of Administrative Sciences
   Code 54KT
   Naval Postgraduate School
   Monterey, CA 93943-5002

5. Commander Naval Surface Forces, Atlantic — 1
   attn: CDR McMickan
   Code N76
   Norfolk, VA 23511

6. Commander-in-Chief Atlantic Fleet — 1
   attn: Mr. Joe Nowlan
   Code N641A
   Norfolk, VA 23511

7. Capt. Andrew Gjelsteen, USA — 1
   Aviation Medicine Clinic
   Silas B. Hayes Army Hospital
   Fort Ord, CA 93941-5800

8. Chairman, Code 52 — 1
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943-5002

9. Defense Technical Information Center — 2
Cameron Station
Alexandria, VA 22304-6145

10. Magdi Kamel — 1
Department of Administrative Sciences
Code 54KM
Naval Postgraduate School
Monterey, CA 93943-5002

11. Chairman, Code 54 — 1
Department of Administrative Sciences
Naval Postgraduate School
Monterey, CA 93943-5002