AFIT/GSM/LSY/92S-1

RANGE SCHEDULING AUTOMATION FOR THE AIR FORCE SATELLITE CONTROL NETWORK: A CASE STUDY IN COMPUTER SYSTEM DEVELOPMENT

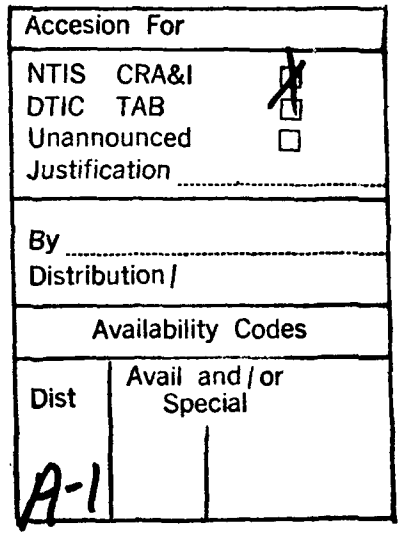
THESIS

Donald J. Aitken, Steven C. Bishop

AFIT/GSM/LSY/92S-1

Approved for public release; distribution unlimited

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 U.S. Government.



AFIT/GSM/LSY/92S-1

RANGE SCHEDULING AUTOMATION FOR THE AIR FORCE SATELLITE CONTROL NETWORK: A CASE STUDY IN COMPUTER SYSTEM DEVELOPMENT

THESIS

Presented to the Faculty of the School of Systems and Logistics

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Systems Management

Donald J. Aitken, B.S.

Steven C. Bishop, B.S.

September 1992

Acknowledgements

We would like to express our sincere gratitude to our faculty advisors, Lieutenant Colonels Chris Arnold and Bill Schneider, for their guidance and support through the metamorphosis of this thesis effort. We appreciate the latitude afforded to us in our research design and the abundance of software and reference materials provided for our use. We would also like to thank Colonel Larry Whipple, Program Director, Satellite Control and Data Handling System Program Office, Headquarters Space and Missile Center for allowing us to collect data from the program offices and the associated program office contractors. Data in information systems studies is difficult to come by, so we especially appreciate his consent and genuine interest in our thesis. We are indebted to several other key "individuals" for providing documentation and then filling gaps in it through interviews and discussions. They are Mr. John List of Paramax Corporation, Mr. Rick Walker of IBM, Mr. Bruce Simpson of The Aerospace Corporation, Mr. Francis Wong of AFSPACECOM and Mr. Walt Danielson of Logicon/Ultrasystems. Finally, we would like to thank our wives and families, Sonia and Kyle Aitken and Rose and Craig Bishop, for their support and patience throughout our effort.

Table of Contents

List of Figures

List of Tables

List of Acronyms

AFB	Air Force Base
AFSCN	Air Force Satellite Control Network
AFSPACECOM	Air Force Space Command
ARF	ASTRO Report Form
ASTM	American Standards for Testing and Material
ASTRO	Automated Scheduling Tool for Range Operations
AUTODIN	Automatic Digital Information Network
BASCH	BAsic SCHeduling
CCS	Command and Control Segment
CCSE	Command and Control Sustaining Engineering
CPCI	Computer Program Configuration Item
CRT	Cathode Ray Tube
CSOC	Consolidated Space Operations Center
DOD	Department of Defense
DSM	Data System Modernization
DTG	Date/Time Group
ECP	Engineering Change Proposal
FQT	Functional Qualification Test
GPS	Global Positioning System
GTS	Guam Tracking Station
HTS	Hawaii Tracking Station
IBM	International Business Machines, Inc.
JCL	Job Control Language
LAN	Local Area Network
MCC	Mission Control Complex
MS-DOS	Microsoft Disk Operating System
MSC	Manning Schedule Change
NASA	National Aeronautics and Space Administration
NCS	Network Control Segment
PAP	Program Action Plan
PC	Personal Computer
RCC	Range Control Complex
RFI	Radio Frequency Interference
RTS	Remote Tracking Station
S/DR	Software Deficiency Report
SAT	Satellite Acquisition Tape
SATAF	Site Activation Task Force
SCRABL	SCheduling Resource Allocation Buffer Linkage
SDC	System Development Corporation
SDLC	Software Development Life Cycle

SLOC	Software Lines of Code
SOC	Space Operations Center
SORD	Statement of Operational Requirements Document
SPO	System Program Office
SRD	System Requirements Document
SSD/CW	Space System Division/ Satellite Control and Data Handling System Program Office
VTS	Vandenberg Tracking Station

Abstract

This study explored the factors influencing divergent outcomes of two computer system development efforts which were undertaken to fulfill the same requirement for computer automation of a manual resource scheduling process. The first project employed the traditional waterfall approach to system development, but resulted in user rejection and cancellation after considerable resources and effort had been expended. The second project employed prototyping and both the process and the product were well received by the users and ultimately produced an operational system. Analysis yielded eight contributory factors to the failure of the first effort. Three of these were related to the waterfall approach, but the remaining five would have adversely affected any type of development effort. As a result, the waterfall approach was not deemed to be the most significant contributor to the failure. However, the major contributor to the success of the second effort was the use of prototyping. Most theoretical advantages of prototyping over the waterfall approach were observed in that effort and two additional advantages were identified. Prototyping's disadvantages were largely mitigated by strong management control of the development process.

RANGE SCHEDULING AUTOMATION FOR THE AIR FORCE SATELLITE CONTROL NETWORK: A CASE STUDY IN COMPUTER SYSTEM DEVELOPMENT

I. Introduction

Judging from the researchers' experience, the mission critical computer resources acquisition process in the Air Force often does not provide products to users in an efficient, effective, and timely manner. Many authors have cited recent development projects, both inside and outside the government, where the final delivered product did not meet all of the users' expectations and typically exceeded budget and schedule constraints as well. The first attempt at automating the Air Force Satellite Control Network's (AFSCN) range resource scheduling function is one such project. After expending considerable resources and time, the resulting system had significant deficiencies and was never accepted by the users for operations. Novel development approaches in a second attempt resulted in overwhelming project success and users' satisfaction. The second system was recently installed and is currently supporting daily operations.

The objectives of this study are to explore the principal differences in the two system development efforts in the context of the unique AFSCN range resource scheduling problem domain, and to understand why the second project succeeded after the first project ended in failure. The case

presents a unique opportunity for comparison of the requirements definition and system development approaches used because the nature of the problem did not change. Only the fundamental problem solving methods changed, within contractual and technological limitations at the time of each development effort. Theory suggests that, in cases like this, use of the evolutionary prototyping method is preferred and contributed greatly to the success of the second project.

To fully comprehend the operational environment for which the two range scheduling automation projects were developed, it is important to understand the AFSCN range scheduling domain. Scheduling operations were a labor intensive process which involved the manual scheduling of satellite supports on the paper acquisition chart. The paper chart provided the master schedule for all AFSCN network resources and allocated these resources to individual satellites for a specified time period. To support scheduling on the paper chart, scheduling personnel also used a computer system to maintain an "electronic" database of the master schedule. The system, known as SCRABL II (SCheduling Resource Allocation Buffer Linkage II), provided the schedulers the ability to cross-check the paper chart with the database prior to disseminating the formal schedule to the various AFSCN organizations. Appendix A provides a more detailed description of the scheduling activities and Appendix B contains a facsimile of the paper acquisition chart. Scheduling operations will be discussed in greater detail in Chapter IV.

The first system aimed at fulfilling the range scheduling automation requirement is called BASCH, for BAsic SCHeduling, and was begun in 1983. In keeping with the accepted practices for Air Force embedded software, the BASCH developer choose to apply the principles of the traditional waterfall model to the project. The developer selected the traditional waterfall model since it provided a structured systems engineering approach. In addition, the Air Force directed the developer to conduct the project in accordance with Military Standard-483, Configuration Management Practices, MIL-STD-490, Specification Practices, and MIL-STD-1521, Technical Reviews and Audits. The BASCH effort will be described in detail in Chapter IV.

The second system is called ASTRO, for Automated Scheduling Tools for Range Operations. The ASTRO system development effort was begun in 1987 after it was clear that the BASCH system would not support operations. In contrast to BASCH, where the developer was directed to use a formalized process of specifications, configuration management, and reviews, the Air Force provided informal direction to the ASTRO developer. Thus, instead of using the traditional waterfall model, the developer chose the evolutionary prototyping approach to meet the AFSCN range scheduling requirements. The ASTRO project will also be described in detail in Chapter IV.

Investigative Questions

The objective of the research is to determine if the results observed in the BASCH and ASTRO projects support the claimed advantages and disadvantages of the evolutionary prototyping methodology as compared to the traditional waterfall methodology found in the literature. To achieve this objective, the following areas of inquiry must be investigated:

1) What is the problem domain? Were the high level system requirements and objectives the same for both efforts?

2) What development methodology was used in the BASCH effort? Why was the methodology chosen?

3) What are the factors cited for the failure of BASCH?

4) What development methodology was used in the ASTRO effort? Why was the methodology chosen?

5) What are the factors cited for the success of ASTRO?

6) What theoretical advantages and disadvantages of the evolutionary prototyping methodology were evident in the ASTRO development?

7) What other factors might have influenced the contrasting outcomes of the two efforts?

The literature review in Chapter II discusses the software acquisition problem in the Department of Defense (DOD) environment and relevant software development methodologies, their origins, and the advantages and disadvantages of their use. The research methodology is presented in Chapter III. A chronological background of the two projects is presented in Chapter IV. Chapter IV also addresses investigative question 1 with a discussion of the range resource scheduling problem, with additional detail

provided in Appendix A. The remaining investigative questions are addressed explicitly in the analysis and findings in Chapter V. Chapter VI presents the conclusions reached through this study.

Concept Definitions

To avoid confusion, traditional software development and prototyping methodologies are defined for the purposes of this study.

Traditional software development methods are the formal, well-defined processes used to develop computer system software. Air Force software development efforts are now predominantly conducted in accordance with DOD Standard-2167A, Defense System Software Development. DOD-STD-2167A establishes procedures used to control development and provides guidance for formal documentation that describes the process and products. Additionally, the standard provides detailed guidance for the management process including configuration management, product evaluation, informal and formal qualification testing, and documentation (Marciniak and Reifer, 1990:37). DOD-STD-2167A defines eight steps in the software development life cycle (SDLC): 1) System Requirements Analysis, 2) Software Requirements Analysis, 3) Preliminary Design, 4) Detailed Design, 5) Software Coding and Unit Testing, 6) Computer Software Component Integration and Testing, 7) Computer System Configuration Item Testing, and 8) System Integration and Testing (DOD-STD-2167A, 1988:9).

The steps identified by DOD-STD-2167A are comparable to the steps associated with the traditional waterfall model. The waterfall model is a structured approach where a series of well-defined, well-documented, and sequential steps are performed to accomplish a system development (Weinberg, 1991:47). Davis, Bersoff, and Comer provide an updated description of the waterfall model in their comparison of SDLC models, as shown in Figure 1. The model now includes feedback loops from subsequent phases of the process (Davis and others, 1988:1453).

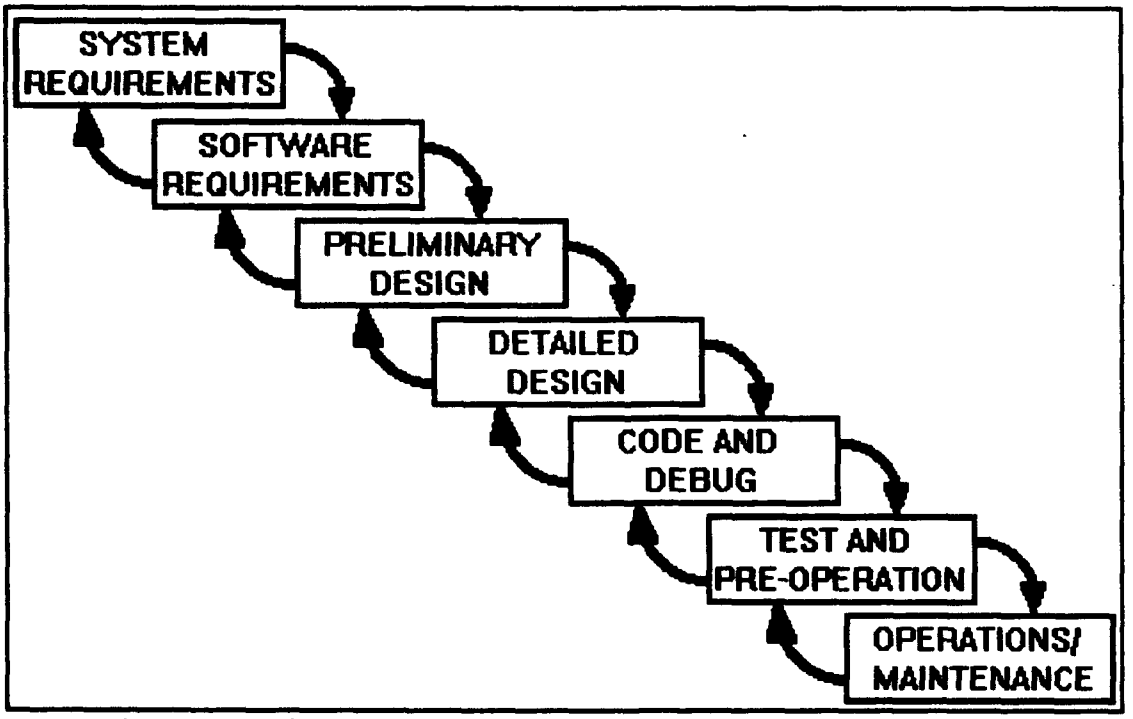


Figure 1. The Waterfall Model (Davis and others, 1988:1453)

Several software development methodologies and tools have been promulgated over the past decade in an attempt to enhance the traditional

SDLC model of software development. One such methodology is software prototyping, analogous to systems engineering (hardware) prototyping which has long been an accepted design aid (Weinberg, 1991:48-49). For example, aircraft and automobile manufacturers have been building prototypes for years to help convert engineers' ideas into reality (Swift, 1989:14). For the purposes of this study, prototyping is an iterative process where successive prototype models are quickly developed and evaluated against the users' requirements. Initially, the developer and users work together to identify and clarify the requirements and specifications for the system. The developer then constructs models of the system which are evaluated by the users. The users' evaluations are fed back to the developer and changes are made in the prototype. This process continues until the users determine that the prototype successfully identifies the critical requirements of the envisioned system. In most cases, the last prototype model will not exhibit all functional characteristics of the operational system but serves as an aid for the final development (Lugi, 1989:13).

Despite its increasing popularity, software prototyping is not a panacea. Attempts at applying the methodology wholesale in lieu of the formal traditional approach, prescribed by DOD-STD-2167A, have not generally improved the final results (Kelly and Neetz, 1988:644). However, in certain types of software development projects, and under certain conditions, software prototyping has been demonstrated to be very effective for defining system requirements and functional capabilities (Voltmer,

1989:24; Carey and Currey, 1989:29; Weinberg, 1991:47). These project types and conditions are not well defined, so managers presently make largely subjective decisions about using software prototyping in their projects. Often, decisions are based upon anecdotal or experiential information which is prolific in information systems literature.

II. Literature Review

A review of literature on current software development methodologies is presented to establish a better understanding of the advantages and disadvantages of the various alternative approaches for the development of software systems within the Air Force and to observe if these advantages and disadvantages relate to the development efforts investigated. The review examined the current problems facing the DOD associated with the inability of the traditional software development methodology, the waterfall model, to deliver quality products on schedule and within budget. The literature review also focused on the applicability of evolutionary prototyping as a means of defining and clarifying users' requirements for computer systems. Finally, the review examined recent research efforts which investigated alternative methods of software development for relevancy to this research effort.

The Software Dilemma

As advances in technology increase the complexity of modern weapon systems, computers are called upon more frequently to control these systems. To better support these complex weapon systems, the computers also have become more complex and now play an integral role in virtually all major systems. A system's performance is dictated by how well the

computer programs can analyze the inputs and tell the system what to do and how to do it (Canan, 1986:46).

In these times of decreasing military budgets, the Air Force must consider all alternatives before committing to the development of a new weapon system. Alternatives, such as modifying existing systems, have become more prevalent in the current acquisition environment. A large portion of these modifications include enhancing embedded computer software to improve the capabilities of the system and to meet new threats (Canan, 1986:46). However, the DOD faces a growing problem associated with software procurement: the inability of the government managers to buy or develop quality software systems in a timely and cost-effective manner (U.S. Congress, 1989:cover letter).

Analysts estimate that the DOD currently spends in excess of $30 billion annually in the development and support of software systems (Staff Study, 1991:1). Within the Air Force, it is estimated that 10 percent of the total budget is allocated to developing and maintaining software (Canan, 1986:46). The 1989 Staff Study from the Subcommittee on Investigation and Oversight, United States House of Representatives Committee on Science, Space and Technology, to the Congress reported that "computer software, which is now a major cost item in many procurements, is not immune from traditional procurement problems such as delays, cost overruns, and poor performance" (U.S. Congress, 1989:3). The report also implied that often times the final software system does not meet the needs

or requirements of the system users. This results in increased expenditures to correct development deficiencies during the software maintenance/support phase which in turn increases the overall life cycle cost of the system.

The traditional approach of specifying, designing, coding, documenting, and testing software does not provide adequate insight into issues that impact software maintenance. The problems associated with the traditional approach become evident when examining the full life cycle cost of software (Figure 2). In analyzing Figure 2, it is noted that approximately sixty percent of the cost associated with the development of computer software is related to maintenance activities. Those activities include: enhancing the performance or maintainability of the software--preventive; modifying the software to support a new processing or data environment--adaptive; and fixing identified processing, performance, or implementation deficiencies--corrective (Boehm, 1981:54-55). For example,

> It cost $85 million to develop the software for an F-16D. It costs another $250 million to maintain that software--rectify its errors, keep it in shape, update it--over its anticipated operational lifetime. (Canan, 1986:49)

Despite the relatively large costs associated with maintenance activities, attention remains focused on creating new tools and alternative methodologies for software design and development. The ease and cost of software maintenance is directly related to activities associated with the software development. Yet, in the traditional approach, there exists very

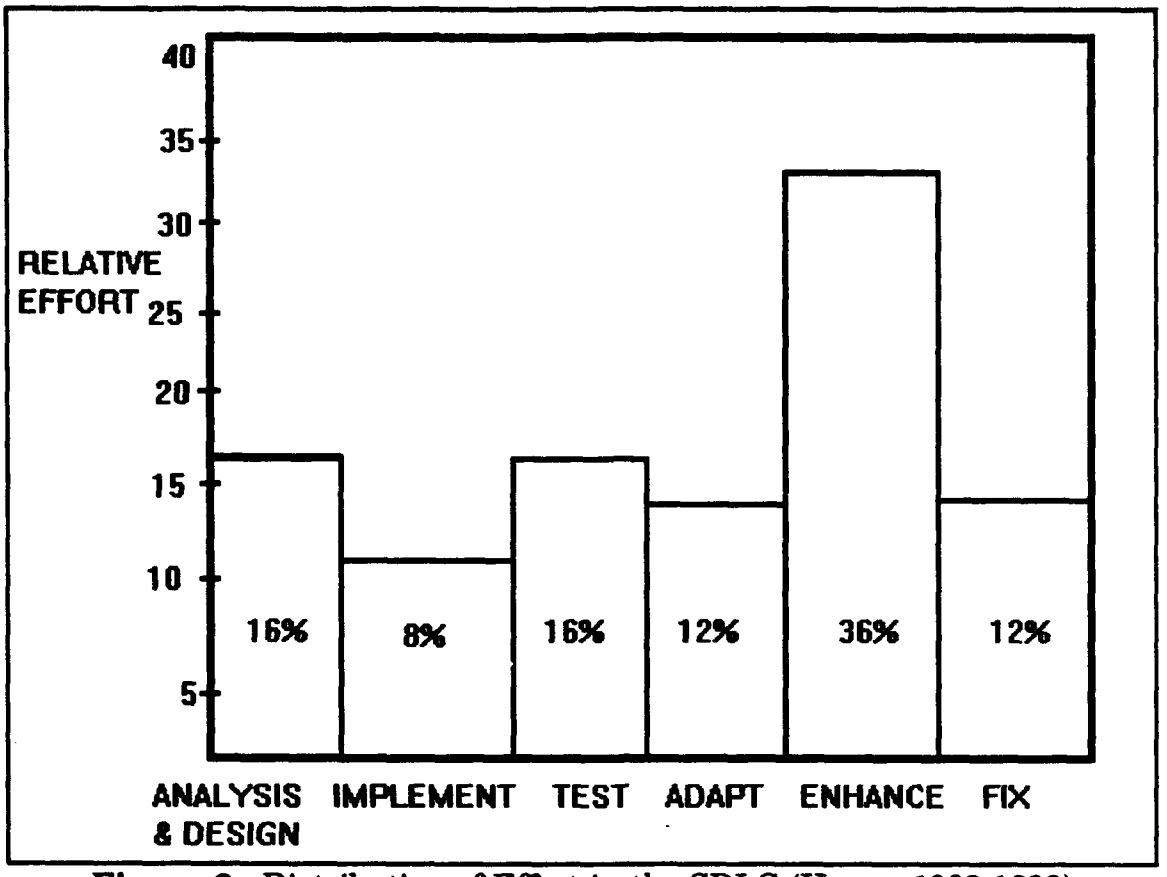


Figure 2. Distribution of Effort in the SDLC (Hager, 1989:1639)

little feedback between the developer and the maintainer. This results in a product that is difficult to maintain and often does not meet the specified requirements (Hager, 1989:1648). Thus, maintenance personnel are left with the responsibility of correcting development deficiencies instead of improving the system capabilities to meet evolving users' requirements.

Traditional Software Development

The traditional approach is a formalized document-driven process where each step ends with a specific deliverable that is then used in the

next step. The process begins with a thorough analysis by the developer, focusing on the scope and feasibility of the system. During the next stage, requirements specification, the users and developer prepare a detailed set of requirements for the system via system-level specifications. These specifications typically include a detailed description of the functional capabilities of the system as well as performance parameters and reliability and physical constraints (Weinberg, 1991:47-48). The specifications provide the foundation for all subsequent phases of the effort.

Upon validation of the system specifications, preliminary and detailed design phases follow. The goal of these phases is to produce a detailed logical and physical description of the system from which the programmers can proceed. The design details data processing requirements, user interfaces, databases, system outputs, and external interfaces. Additionally, the system hardware on which the software will run is identified (Weinberg, 1991:48).

After approval of the detailed design, the developer enters coding and testing phases. During these phases, computer software is developed or purchased and testing of both individual software subsystems and the overall system is conducted. Ideally, if the system specifications were complete and the design well thought-out, the actual development and test can be reasonably uncomplicated. However, many times the specifications and design are less than complete, which results in errors in the functional capabilities. Thus, errors in requirements specification are usually not

identified until late in the development and are extremely costly to correct (Weinberg, 1991:48).

Furthermore, the later the requirement error is discovered, the more impact the correction has on the development schedule and the overall life cycle cost (U.S. Congress, 1989:9; Boehm, 1981:40). For example, if a software requirement error is detected during the requirements phase, its correction has minimal impact on the development effort. In contrast, if the same error is not discovered until after system delivery, the correction involves a significant effort at substantial cost. Past research has estimated the cost to fix an error during the maintenance phase instead of during the requirements phase at 100 times greater (see Figure 3).

For Air Force projects, the traditional approach begins with the users submitting their requirements in a System Requirements Document (SRD) for a new system or modification of an existing system. In theory, the SRD outlines the users' expectations of what the system must do and not how to actually implement the requirements. The SRD provides a starting point for all subsequent design and development activities and serves as the criteria for system validation (Rowens, 1990:12).

Following submittal of the SRD, the developer presents the design via the system specifications. The correctness of these specifications often depends on how well the users understand the technical aspects of the project and whether they can visualize the results from the technical descriptions (Swift, 1989:14). Thus, the success of the entire project hinges

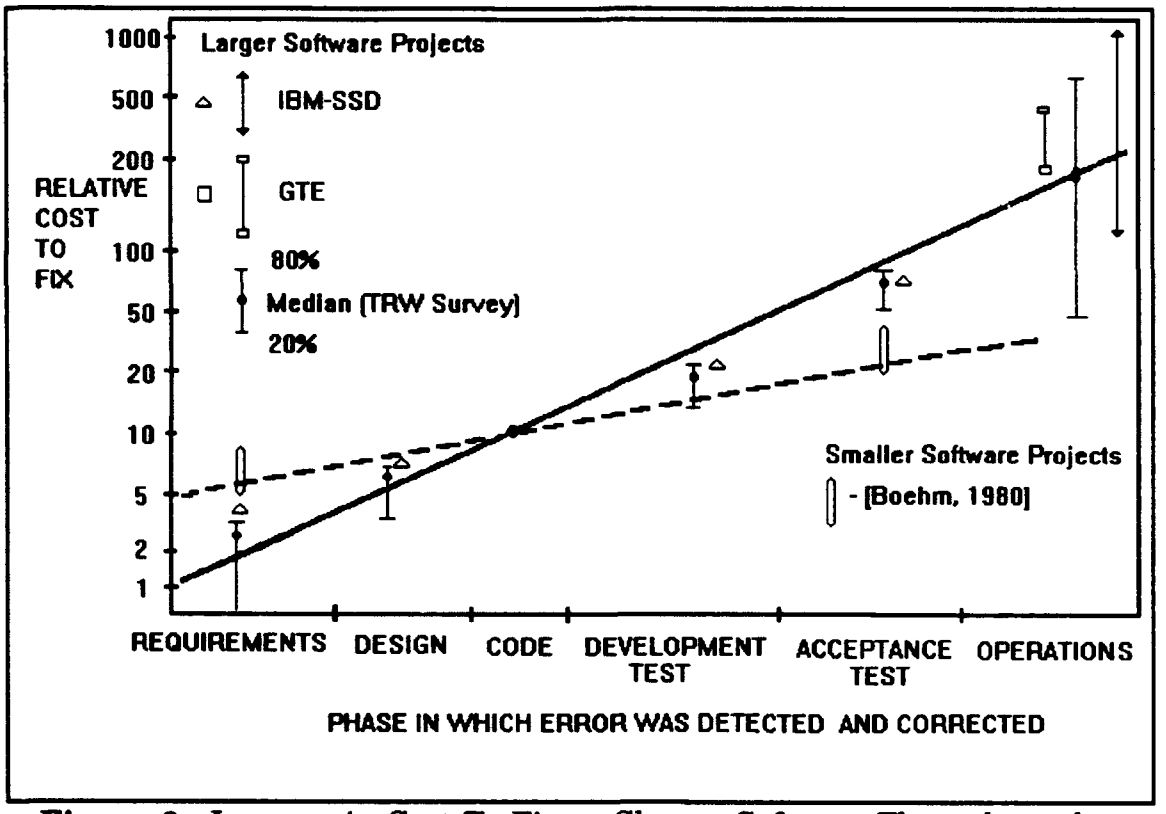


Figure 3. Increase in Cost-To-Fix or Change Software Throughout the Life Cycle (Boehm, 1981:40)

on the accuracy of the requirements analysis and system design. The formal stages of the traditional approach, when explicitly followed, should produce a computer system that meets the specified requirements (Gavurin, 1991:13). However, the traditional approach has been ineffective in meeting the requirements for development of complex interactive systems required for DOD weapon systems (Canan, 1986:46-48). Many of the problems associated with the traditional approach can be tied to two areas; 1) the users' inability to clearly and accurately convey their requirements via formal specifications early in the process, and 2) the associated time span

between original requirements definition and delivery of the final system (Gavurin, 1991:13).

In 1987, The Defense Science Board Task Force on Military Software reported that military software is procured because of the inherent flexibility it provides modern weapon systems. Nonetheless, the DOD acquires these critical software systems under an acquisition approach and regulations which demand extreme rigidity on the part of the developer (U.S. Congress, 1989:8). The Task Force added:

> The most common present method of formulating specifications--issuing a Request for Proposal, accepting bids, and then letting a contract for software delivery--is not keeping with good modern practice and accounts for much of the mismatch between users' needs and delivered function, cost and schedule. [We] now understand the importance of iterative development of requirements, the testing of requirements against real users' needs by rapid prototyping, and the construction of systems by incremental development, with early releases subject to operational use. (U.S. Congress, 1989:8)

Additionally, the Task Force report suggested the DOD change its development standards to remove the remaining dependency on the waterfall approach and to institutionalize the use of prototyping and incremental development. As a result of the Task Force's recommendations, the DOD adopted DOD-STD-2167A as the new standard for software development. However, in response to issuance of the new standard the report stated:

> Although some parts of DOD-STD-2167A appear to encourage modern development methods, the document as a whole continues to reinforce the document-driven, specify-then-build

> approach that we believe causes so many of DOD's software problems. (U.S. Congress, 1989:8)

The report concluded that fully imposing the complex standards of DOD-STD-2167A on every piece of software purchased by the military would drive the life cycle cost of the system through the roof, would result in a mass of useless paper, and would reduce safety in critical systems by diverting management attention (U.S. Congress, 1989:21).

In some cases the traditional approach does remain attractive to a program manager. The waterfall approach defines specific documents and milestones associated with each development phase, which provide the necessary checkpoints and control mechanisms (Gavurin, 1991:13). At each milestone, the manager can assess the effort and assure the project is progressing as planned.

Despite all these controls, computer systems developed under the traditional approach are often unsuccessful (Swift, 1989:14). Availability of the end system is often delayed due to misinterpretation of the requirements by the developer or changes in the requirements by the users. These delivery delays further frustrate the users, who anxiously await the required system.

In recent years, several new development methodologies have emerged as alternatives to the traditional waterfall approach. The overall goal of these alternatives is to reduce the total life cycle cost of the software by improving the development process and reducing the need for the

maintainers to correct development problems (Hager, 1989:1648). By reducing the demand on programmers to conduct corrective maintenance, more effort can be applied to adaptive or perfective maintenance activities, which in turn will lead to improved system performance. Furthermore, since the alternative methodologies improve the process for identifying system requirements, they also reduce the amount of programmer effort required to perform perfective maintenance as well.

Prototyping

One of the more widely accepted alternatives to the traditional development approach is software prototyping. Prototyping offers an approach to designing and developing computer systems that minimizes the problems and risks inherent in the traditional approach.

Prototyping differs from the traditional approach in two major ways. First, the prototype system most often does not include all the functional capabilities required for the final system, nor is it expected to have the same performance characteristics required to meet the users' processing needs. Second, the prototype system should be developed quickly through an iterative process and at significantly lower cost than the final system. Michael C. Holloway in his article provides one definition of prototyping:

> A software system that exhibits essential features of an intended system: it is an adequate representation of the intended system in some ways but not in others (similar to a scale model). (Holloway, 1987:1)

Holloway classifies the prototyping approach in one of two ways: by scope or purpose. Scope is the extent to which the prototype models the final system. It may contain the full functionality of the final system, only part of the functionality, only representative functionality, or simply the intended user interface. Purpose is defined as the goals or objectives for which the prototype system is being developed. Prototypes are developed to clarify system requirements, to test design approaches, or incrementally build the final system (Holloway, 1987:2).

One of the primary objectives of the prototyping approach to software development is to get the users involved early in the project to assist them in defining and clarifying the system's requirements. This is accomplished by combining the systems analysis, requirements analysis, preliminary design, and detailed design stages of the traditional approach to minimize initial design and development time. As a result, the users get earlier insight into the system and they are able to evaluate the prototype to assure it meets their needs.

To this end, the prototyping process consists of numerous iterations of the following four fundamental steps (see Figure 4):

1) Identify the users' initial requirements.

2) Develop an initial prototype system.

3) Users evaluate the prototype.

4) Revise and enhance the prototype system based on users' feedback. (Er, 1987:13)

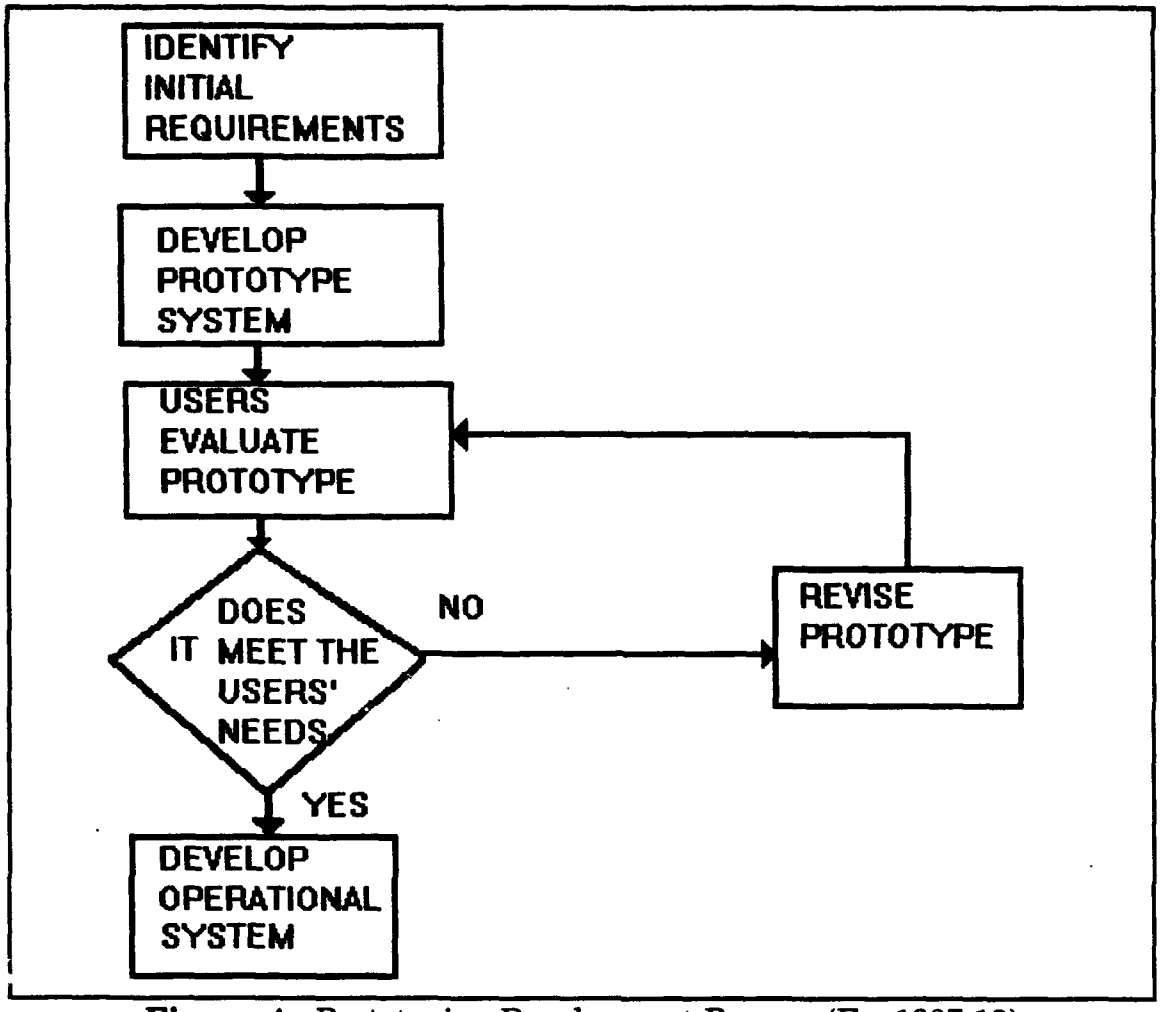


Figure 4. Prototyping Development Process (Er, 1987:13)

Through this iterative process, prototyping provides a means to develop a system more quickly and to define requirements more accurately (Carey and Currey, 1989:30).

The concept of prototyping is not foreign to the Air Force. In fact, Air Force Regulation (AFR) 800-14, Acquisition Management: Life Cycle Management of Computer Resources in Systems, defines the process for applying the prototyping methodology to software projects:

> Initial requirements definition and design are abbreviated. The initial system is developed quickly to meet initial functional requirements, but may not meet performance requirements such as speed or capacity. User experience with the prototype is fed back into requirements for a deliverable product which is developed using more traditional techniques. (AFR 800-14, 1986:26)

However, the regulation explicitly directs the use of prototyping be limited to only the requirements definition phase. Yet, recent efforts in both the DOD and the commercial sector have shown that prototyping can be successfully applied throughout the development process especially during design phases.

Prototyping Advantages. Discussions so far have focused on the benefits of prototyping in general terms. Several specific benefits are derived when applying prototyping techniques.

Possibly the two greatest benefits of prototyping lie in the areas of requirements definition and users' involvement (Carey and Currey, 1989:30; Gavurin, 1991:14). Since prototyping consists of an iterative process between development and users' evaluation, the users can provide valuable feedback to the developer. This feedback tells the developer early on how well the system meets users' requirements (Holloway, 1987:2). In the traditional approach, the system users must provide a comprehensive list of requirements up front. Unfortunately for new ground-breaking systems, requirements analysts are often told by the users, "I'm not really sure what I want, but I'll know it when I see it" (Boehm, 1984:290). Prototyping also provides faster response times to the users' changing requirements (Swift,

1989:15). New requirements can be directed back to the developer through the feedback channels and swiftly incorporated into the prototype system. Furthermore, prototyping provides a useful tool in allowing the developer to assess users' feedback regarding various design alternatives. Through the application of prototyping, the users and the developer have a clearer understanding of what is being produced, which results in fewer surprises (Smyrniotis, 1990:202). Thus, prototypes offer a greater probability of delivering what the users need with the initial operational delivery (Swift, 1989:15).

In addition, prototyping can be used to stabilize the requirements for both new systems and proposed modifications to existing systems. Feedback through users' evaluations is essential for effectively validating complex requirements found in large embedded real-time computer systems (Lugi, 1989:24).

Another significant benefit of prototyping is the ability to minimize the risk and uncertainty associated with the development effort. Purtilo and others note that by reducing uncertainty, the developer minimizes the risk that incomplete requirements would lead to system failure (Purtilo and others, 1991:3). For DOD projects, the program manager is responsible for controlling all aspects of the project, including minimizing cost, schedule, and performance risks. DOD Directive 5000.1, Defense Acquisition, Part 1, Section C, recommends that "technology demonstration and aggressive prototyping (including manufacturing processes, hardware and software

systems, and critical subsystems) coupled with early assessment, are to be used to reduce risk" (DOD Directive 5000.1, 1991:4-5). Prototyping reduces the risk associated with the project by identifying problems early in the process and providing more time for the developer to resolve them.

Finally, prototyping is useful in developing user interfaces for systems with numerous end users. Susan Harker writes:

> In the development of large-scale applications, those involving large quantities of data, complex processing and many end users, the implications of creating a poor user interface are particularly serious. The scale of development often makes it difficult to correct problems which derive from the fundamental decisions about the nature of the system and the numbers of people affected results in higher costs to run the system because of potential errors and lower efficiency. (Harker, 1988:420)

Harker suggests that prototyping can be applied to provide "concrete representation" of a possible solution on which the users can comment directly. Therefore, the developer can ensure that all possible steps have been taken to provide a user interface which does represent a good solution for the users (Harker, 1988:420).

The findings of the literature review suggest that prototyping offers numerous advantages over the traditional waterfall model. Those advantages are summarized in Table 1.

Prototyping Disadvantages. As with all development processes, there are potential disadvantages that a prototyping project manager must be aware of before undertaking the effort. Unlike the traditional waterfall approach, prototyping does not provide clear and formalized deliverables,

Table 1.

Advantages of Prototyping (Kelly and Neetz, 1988:644; ASTM Standard, 1990:413)

1.	Provides a means for users and developer to work together defining and developing system.
2.	Provides greater insight into users' requirements.
3.	Prototypes can be easily modified to accommodate changing users' requirements.
4.	End product is a more stable, tangible representation of the users' requirements.
5.	Provides a better estimate of the time and effort required to develop the final system.
6.	Provides early visibility and elimination of development problems.
7.	Reduces development risk and uncertainty.
8.	Can quickly produce a working model to provide a clearer understanding of system's behavior.
9.	Allows flexibility in selection of hardware and support software for the operational system.
10.	Can produce a better user-system interface for large multi-user systems.
11.	Provides higher probability of delivering the "right" system the first time.

which would provide management insight into the project. Therefore, attempts to apply traditional controls to prototype efforts often result in feelings of uneasiness among management (Voltmer, 1989:24). Given the

lack of a formalized process, management perceives the project is out of control and may decide to cancel the project. To avoid this problem, management must be made aware of the differences between the two approaches and alternative checkpoints and milestones must be developed.

In addition to maintaining overall management control, the developer must maintain strict control of the project requirements process. Since most prototyping projects forego the formal requirements process, the users often do not find time to completely analyze the system requirements until the prototype effort nears completion. In a flurry of activity, the users realize the "window of change" is closing and the developer often receives a flood of new requirements near the end of the project (Swift, 1989:19). Studies have shown that in prototyping projects as much as 50 percent of the system's functional requirements may not show up until the last 10 percent of the development schedule (Voltmer, 1989:25). Therefore, the developer must be aware of this situation and maintain strict control over system requirements growth.

Another potential drawback faced by prototype project managers stems from an inherent strength of the process (Voltmer, 1989:25). Since prototype systems are much easier to change than traditional systems, there may be a desire by the users to continually request minor changes and improvements. The developer must realize that eventually the prototype effort must be terminated and the project completed. If the developer allows the project to continue, he may find the computer system

in a perpetual state of 99 percent complete with no end in sight (Voltmer, 1989:25).

Moreover, the application of prototyping techniques may be impractical for certain projects. Purtilo, Larson, and Clarke discuss the potential problems associated with using prototyping for large-scale development efforts.

> Many large-scale prototyping activities are motivated by applications that must incorporate parallelism or distribution within the implementation. Unfortunately, when programmers must manage multiple threads of control, or implement communications protocols, their programs become difficult and time consuming. That is, the prototype ceases to exhibit any economic benefits to the developer. (Purtilo and others, 1991:3)

Finally, prototyping may negatively affect the performance and adaptability of the modeled system. Since one of the goals of prototyping is to quickly generate an operational model of the system, performance is often overlooked during the development. The prototyping effort may result in an inefficient system and when pushed to its maximum levels (e.g., heavily loaded, buffers exhausted, and displays filled with data), the system may not perform up to the required level of performance. Furthermore, often times the prototype may not be useable outside of the hardware and software environment in which it is developed. Thus, the prototype would require significant redevelopment to make it portable to a more desirable operational system (ASTM Standard, 1990:413).

In summary, the literature suggests there are numerous potential disadvantages associated with the use of prototyping during the development effort. Table 2 lists those disadvantages.

Table 2.

Disadvantages of Prototyping (ASTM Standard, 1990:413)

1.	Complete requirements definition often occurs late in the development process.
2.	Process lends itself to growth in system requirements through iterative users' feedback.
3.	May be slow to identify an optimum design.
4.	May result in an inefficient system design.
5.	Potential risk of the prototype being forced into serving as the operational system.
6.	May produce a system which lacks portability or generality.
7.	Not practical for large-scale developments which apply parallelism or distribution of effort.
8.	Can be difficult to maintain a tight development schedule.
9.	Does not provide clear, formalized deliverables for management to track development progress.
10.	Documentation tends to lag behind development.

Prototyping Approaches. The literature divides prototyping efforts into two distinct categories, throw-away and evolutionary (Gavurin, 1991:13; Guimaraes, 1987:102; Rowen, 1990:14). Each approach has different

applicability in software development and selection of an approach depends on the specific project.

Proponents of throw-away prototyping maintain that a prototype system should be considered only as a temporary system, and the final system should be built from scratch using traditional methods (Rogers, 1986:3). Successive throw-away prototypes benefit from the experience gained with each effort. Throw-away prototypes are extremely useful when the developer is unable to define a clear design alternative. In this case, the developer builds throw-away models of particular design alternatives to assess the feasibility of the design (Guimaraes, 1987:103; Holloway, 1987:2). With each model, the developer tests system characteristics (e.g., system response time, memory requirements) for various design alternatives. These prototypes are far less costly than proceeding into development and finding the wrong system configuration has been specified. Thus, prototyping in the context of the throw-away model is a method of system analysis that involves elaborate simulation of system features and intense involvement of the users (Klinger, 1986:131).

In determining if throw-away prototyping is applicable for a specific project, the developer must consider duplication of effort between the prototype and the final system. Throw-away prototypes often waste valuable hours in building the initial system only to turn it over to the final development group, who extract the necessary information on the users' requirements, discard the code, and begin the development from scratch

(Guimaraes, 1987:102). For example, a Chicago bank used throw-away prototypes to support internal software requirements. On average, the bank expended 250 hours per prototype. An additional 45 hours were expended by the prototype developers in discussing the project requirements with the final developers. The system development group took between 75 and 225 hours repeating work already done under the prototype effort. The result was 30 to 90 percent of the development effort expended being redundant activity (Guimaraes, 1987:102).

In deciding whether to use throw-away prototyping, a manager must consider the development schedule. Throw-away prototyping usually requires more development time as compared to the evolutionary approach because each subsequent model starts from scratch (Rogers, 1986:3). Thus, in those cases where time is not of the essence, throw-away prototyping offers an attractive development alternative.

In operational terms, evolutionary prototyping is the process of continually gathering knowledge regarding the system's requirements and converting that knowledge into operational software until such time as the prototype meets the users' needs. The knowledge sought may pertain to the definition or clarification of requirements, users acceptance, system characteristics or performance factors (Tate, 1990:238).

In evolutionary prototyping, the final system evolves directly from the prototype model through continuous development refinement. Successive models build on experience gained from the previous iteration. Upon

completion of each model, the developer delivers the prototype system to the users for evaluation. As the users evaluate the prototype, they communicate to the developer any required modifications. The prototype is again modified and a new model is delivered to the users for evaluation. The iteration process continues until the point where the developer and users agree the prototype system meets the users' needs.

Once the prototype meets the users' requirements, the process is discontinued and the model becomes the baseline for the final operational system. Additional development actions may be required and range from the complex actions of recoding the software into a more supportable language, to rehosting the system and defining new hardware to meet performance parameters, or to the simple action of producing documentation of the system "as-designed." In these cases, the prototype evolves into the final system (Gavurin, 1991:14).

Finally, at the start of the prototyping process no system specification is needed, since evolutionary prototyping allows the system to change based on users' feedback. The prototyping process will ultimately produce the system specification.

Incremental Development

The incremental development model divides large, complex software projects into manageable subsystems. These subsystems are then developed in parallel as individual "mini" projects. Individual components are then

integrated into a single increment, which provides a "snapshot" of the software system at any given point in time. The developer can use these increments to assess the overall progress of the project.

In addition, the entire software system can be delivered as a series of increments which are integrated together over time. These increments provide a limited software capability to the system users, while other more complex and time consuming subsystems are developed. Incremental development can also reduce the overall schedule, since each increment can be tested separately prior to integration. Precise configuration control must be maintained to assure that problems with one increment do not result in the failure of another. Therefore, management must maintain strict control to assure project success (Marciniak and Reifer, 1990:47).

Incremental development can be used at the system or subsystem level and is often used when a "breadboard" or model is developed prior to production. Similar to prototyping, incremental development stresses the current trend in software engineering to build a system model prior to committing time and resources to production of the operational system.

The Spiral Model

Numerous advanced software development approaches have evolved to take advantage of the benefits of the traditional waterfall model and the modern practices of alternative models. One such approach is the spiral model. The major distinguishing factor of the spiral model is that it creates

a risk-driven approach to software development instead of the primarily document-driven approach of the traditional waterfall model (Boehm, 1988:88). The basic goal of the spiral model is to reduce development risk by using an iterative approach that builds upon the lessons-learned throughout the project (Marciniak and Reifer, 1990:48).

At project initiation, the project manager and software engineers determine the overall development risk. Based on their assessment, the team can tailor the development approach. If the risk is high, an initial prototype may be developed to assist in the further definition of the project objectives. Each subsequent cycle of the spiral begins with the identification of the objectives of the cycle, the alternative means of development to be used during the cycle, and the constraints associated with the selected approach (i.e., cost, schedule or system performance) (Boehm, 1988:92). Additionally, at the start of each cycle, the development team conducts a risk analysis to determine whether to continue the current spiral or jump to another spiral. If the team determines that the project risk is extremely low, they may decide to jump to the final cycle of the model, the traditional waterfall approach.

During the early cycles of the model, documentation and formal management controls are minimized to reduce the development costs. As the model progresses through the various cycles, the level of documentation increases as well as the rigor of the development (Marciniak and Reifer, 1988:48).

The spiral model reflects an underlying concept that each cycle in the spiral follows a progression that addresses the same sequences of steps. These steps are applied to each portion of the development, from the overall project concept all the way down to the coding of individual software subsystems (Boehm, 1988:92).

Prototyping Versus Specifying

In Boehm, Gray, and Seewaldts' article on prototyping versus specifying, the authors describe an experiment which they conducted to confirm and quantify the positive aspects of prototyping as an alternative software development approach. The experiment was conducted to answer the following question: "Should the current specification-driven approach to software development be dropped in favor of an alternative based on prototyping?" (Boehm and others, 1984:290). The authors' motivation was based on recent proposals that suggested prototyping offered a number of advantages, such as the early resolution of high-risk issues and the flexibility to adapt to changing environments and users' needs, over the traditional approach to development (Boehm and others, 1984:290).

The experiment was conducted with students in a first year graduate course in software engineering at the University of California at Los Angeles. Students were divided into seven teams based on personal preference toward developing software by specifications or prototyping. Four teams used a specification-driven approach while the other three

teams used a prototyping approach. Each team was required to develop the same software product (a user-interactive software cost estimation model comprising approximately 3000 Pascal source instructions). The four specification-driven teams had to produce a requirements specification, a design specification, an end product consisting of operational code and users' and maintenance manuals. The three prototyping teams were required to produce the same end products and were required to produce and exercise their prototypes by midpoint of the class (Boehm and others, 1984:291).

The results of the authors' experiment were: 1) prototyping teams on average developed products that were 40 percent smaller and required 45 percent less effort; 2) there was no significant difference in overall productivity between the two groups; 3) overall effort was proportional to product size; 4) overall product performance was generally the same--the prototype products were rated lower in overall functionality and their tolerance for errors in input, but correspondingly higher in the ease of learning and user interface; 5) maintainability of the prototype products was rated higher than the specified products, but the prototyping products were rated lower on the basis of ease of add-on; and 6) the specification teams were more productive in developing documentation (Boehm and others, 1984:291-299).

Based on the results of the experiment, the authors concluded that both prototyping and specifying provide valuable advantages that complement each other. Specifically, for both large and small projects a

mixture of the two approaches would be beneficial and preferable to the exclusive application of either. Finally, they noted that risk management should be the driving factor in determining the specific mix of the two approaches. Each software project should develop a risk management plan which identifies potential high risk issues, establishes plans for dealing with them, and documents risk resolution for project status reviews (Boehm and others, 1984:300).

III. Research Methodology

The guiding research question for this study is: Do the results observed in the BASCH and ASTRO projects support the claimed advantages and disadvantages of the evolutionary prototyping methodology over the traditional waterfall methodology found in the literature? The research methodology establishes the plan for answering this question and the more detailed investigative questions identified in Chapter I.

Research Design

The case research method is employed, with the two AFSCN range scheduling automation efforts as the units of analysis in a single-case embedded design. The events, decisions, methods, impressions, and results of the effort are chronicled to seek explanation for the divergent outcomes of the two development projects. The study is explanatory in nature, to determine if the theoretical advantages of the evolutionary prototyping method explain the outcomes in this case. The research also attempts to determine whether other plausible explanations for the failure of BASCH and the subsequent success of ASTRO can be considered insignificant.

The site was selected because it presents a unique opportunity to compare the traditional method of software development and the evolutionary prototyping method in the same problem context. Also, thorough data collection and detailed contextual development is facilitated

because both researchers previously worked in the organization which managed the projects. The objective of the case study is to determine if application of the evolutionary prototyping methodology in the ASTRO project significantly contributed to its success.

The reasons for selecting the ex-post-facto research design are twofold: 1) success via prototyping has been proclaimed, but there is scant empirical evidence (judging from the literature search) to support the claim and encourage future use; and 2) the time required to conduct an experiment in the population of interest is well beyond the time available for this research. This is because the development phases of the SDLC typically span more than a year. Also, experimental control of extraneous variables and collection of data would probably not be manageable across a sample of contracted development efforts.

Methodology Literature Review

In his book on case study research, Yin provides the following definition:

> A case study is an empirical inquiry that: investigates a contemporary phenomenon within its real-life context; when the boundaries between phenomenon and context are not clearly evident; and in which multiple sources of evidence are used. (Yin, 1989:23)

He emphasizes that case studies are not generalizable to populations or universes, but rather to theories. The case is not a sample in a statistical analysis, and should not be treated as such. It is used "to expand and

generalize theories (analytic generalization) and not to enumerate frequencies (statistical generalization)" (Yin, 1989:21).

In their paper on the use of the case research strategy in information system studies, Benbasat, Goldstein, and Mead attribute recent interest in the case method to "dissatisfaction with the type of research information provided by quantitative techniques." They cite the complexity of multi-variate statistical methods and the assumptions about the underlying distributions, requirements for large sample sizes, and difficulty in understanding and translating results. In their view, "the case strategy is particularly well-suited to [Information System] research because the technology is relatively new and interest has shifted to organizational rather than technical issues" (Benbasat and others, 1987:382). They note that detailed chronological case studies have helped to understand the causes of success or failure in particular information system projects.

Benbasat and others cite three reasons that the case research method is appropriate for information system studies:

> First, the researcher can study information systems in a natural setting, and learn about the state of the art, and generate theories from practice. Second, the case method allows the researcher to answer "how" and "why" questions, that is, to understand the nature and complexity of the processes taking place. Questions such as, "How does a manager effectively introduce new information technologies?" are critical ones for researchers to pursue. Third, a case approach is an appropriate way to research an area in which few previous studies have been carried out. (Benbasat and others, 1987:370)

Design Considerations. The research design guides the research by establishing the questions to be investigated, the relevant data, the subset of data to be collected, and the analysis which links the data to the questions. Yin breaks the research design into five components which are important for case study research. They are: "1) a study's questions; 2) its propositions, if any; 3) its unit(s) of analysis; 4) the logic linking the data to the propositions; and 5) the criteria for interpreting the results" (Yin, 1989:29). The questions for which case studies are appropriate are "how" and "why" type questions. When the purpose is explanation, the proposition helps determine what should be studied (i.e., site selection and unit of analysis). Data analysis and interpretation, addressing the last two components, are the least well developed aspect of case study design (Yin, 1989:29-33).

Yin categorizes case study designs into four types. They are single-case holistic (single unit of analysis), single-case embedded (multiple units of analysis), multiple-case holistic, and multiple-case embedded. Three rationales for selecting the single-case design are provided. The first is if the case is a critical case which can "confirm, challenge, or extend" a well-formulated theory. The outcomes in the case can either be explained by the theory's propositions or by alternative explanations. The second is if the case is so "extreme or unique" that any occurrence of the phenomenon of interest is worth documenting. Finally, the third rationale is if the case is

"revelatory," or previously inaccessible to scientific investigation (Yin, 1989:47-49).

Benbasat and others discuss the research design in terms of the overall themes and objectives. They note in their evaluation of several papers from information systems journals that "the predominant theme in the case studies was implementation, that is, the possible causes of the success or failure of an information or decision support system" (Benbasat and others, 1987:378). Here the case study design is useful because of the long time span involved in implementing an information system, the complexity of the processes, the large number of people involved, and the chance occurrence of events. They also note that in the case studies investigated the objective of the research was not always clearly defined. The researchers did not state, and it was not always clear, whether the study's purpose was exploratory or explanatory (Benbasat and others, 1987:378-382).

Additionally, Benbasat and others discuss practical considerations of single-case versus multiple-case design, unit of analysis, site selection, and data collection methods in the conduct of case studies. They suggest that single-case studies are "most useful at the outset of theory generation and late in theory testing," and "may also be used to test the boundaries of well-formed theory" (Benbasat and others, 1987:373). The unit of analysis can be an individual, group, organization, or a specific project or decision. The research questions and the objective (i.e., to generalize to other individuals,

organizations, or projects) should guide the choice. Site selection is determined by the same factors which led to the single-case design. Lastly, the data collection should be from two or more sources "to obtain a rich set of data surrounding the specific research issue, as well as capturing the contextual complexity" (Benbasat and others, 1987:372-374).

In her article, Eisenhardt notes that cases are chosen for theoretical, not statistical reasons. Single, within case, analyses are appropriate if the intent is to focus on the dynamics involved in a single setting. "The cases may be chosen to replicate previous cases or extend emergent theory, or they may be chosen to fill theoretical categories and provide examples of polar types" (Eisenhardt, 1989:534-537).

Quality Considerations. As in other forms of empirical research, the issues of validity and reliability must be addressed in a case study. Yin discusses four tests for quality of research designs. They are defined as:

> *Construct validity*: establishing correct operational measures for the concepts being studied;
> *Internal validity* (for explanatory or causal studies only, and not for descriptive or exploratory studies): establishing a causal relationship, whereby certain conditions are shown to lead to other conditions, as distinguished from spurious relationships;
> *External validity*: establishing the domain to which a study's findings can be generalized; and
> *Reliability*: demonstrating that the operations of a study--such as the data collection procedures--can be repeated, with the same results. (Yin, 1989:40-41)

Tactics for achieving these objectives in case studies are provided in Table 3.

Table 3.

Case Study Tactics for Four Design Tests (Yin, 1989:41)

Tests	*Case-study Tactic*	*Phase of Research in Which Tactic Occurs*
Construct validity	use multiple sources of evidence	data collection
	establish chain of evidence	data collection
	have key informants review draft case study report	composition
Internal validity	do pattern matching	data analysis
	do explanation-building	data analysis
	do time-series analysis	data analysis
External validity	use replication logic in multiple case studies	research design
Reliability	use case study protocol	data collection
	develop case study database	data collection

Data Collection

The data collected consists of program documentation, archival records, and unstructured interviews with key development and operations personnel. Appendix C contains the interview preface used to conduct each interview. The differing sources of data are used for triangulation within the study to enhance internal validity. Furthermore, "two researchers can

capture greater richness of data and rely more confidently on the accuracy of the data" (Benbasat and others, 1987:374).

Data Analysis

The analysis is separated into two tasks. The first task is to detail the background and events of the case. Chapter IV contains a chronological history of the range resource scheduling automation efforts from the beginnings of BASCH to the operational activation of ASTRO. The second task is to provide answers to the investigative questions through analysis of the data collected. Chapter V contains a narrative analysis and findings for each investigative question.

IV. Case Background

To understand the complexity of the development effort associated with the BASCH and ASTRO projects, it is important to provide some background into the environment in which the computer systems operate. The AFSCN is operated by the Air Force Space Command (AFSPACECOM) and consists of a global network of space and ground tracking, telemetry, commanding, mission operations, and communications resources dedicated to the support of manned and unmanned DOD space programs. Programs supported by the AFSCN include communications, navigational, environmental, and surveillance satellites which provide data to military and other national security agencies. The AFSCN is responsible for providing day-to-day spacecraft operations, which include monitoring health and status of the on-orbit satellites, performing spacecraft anomaly resolution, and ensuring that the satellite's mission data is available for the operational users. The primary role of the AFSCN is to provide the command and control facilities required for satellite launch and on-orbit checkout, and the resources necessary to maintain orbiting DOD satellites so that these vehicles can perform their missions in a timely and effective manner (Staff Study, 1991:i).

To support these spacecraft missions, it is necessary to schedule regular contacts with the satellites using AFSCN network resources. A

contact consists of the transmission of commanding data to the spacecraft and/or the receipt of health and status information and mission payload data. The commanding data directs the spacecraft to perform the necessary maneuvers and activities required to support the vehicle's operational mission. The transmitted data includes the status of on-board spacecraft systems (e.g., fuel levels, battery voltage, subsystem configuration and availability), the orbital location and velocity/direction of the spacecraft, and the requested mission data. The transmitted data is received by one of the AFSCN Remote Tracking Stations (RTSs) and is relayed via a dedicated global communications system back to the AFSCN Mission Control Centers (MCCs) for processing and analysis.

Integral to the AFSCN are the Range Control Complexes (RCCs) located at Falcon Air Force Base, Colorado Springs, Colorado, and Onizuka Air Force Base, Sunnyvale, California. The RCCs are responsible for the scheduling, allocation, and conflict resolution of all AFSCN ground-based resources. The RCCs accomplish their task by evaluating the resource requirements of the various spacecraft users and allocating network resources in the most effective and efficient manner possible to support each operational mission. The output of the RCCs is the AFSCN master schedule which assigns all required network resources to a specific spacecraft for a given period of time.

In the early years of military space programs, the complexity of satellite contact scheduling was sufficiently low that a daily schedule of

satellite contacts could be maintained manually on a paper acquisition chart. The range scheduling acquisition chart has been used to support operational satellite scheduling for more than 30 years. The chart is a tool used to collect requests for resources, identify and resolve resource conflicts, prepare support schedules, and monitor execution of the completed schedule. The acquisition chart in conjunction with a computer database has proven to be an extremely effective, flexible, and reliable tool. In addition, a very sophisticated set of procedures have been developed and refined to support the acquisition chart concept and to provide continuous uninterrupted scheduling of AFSCN spacecraft and ground resources (Unisys, 1987:1). However, with the continued growth in the number, size, and complexity of both ground and space resources, combined with the increased importance these space assets have on national defense, it became necessary to develop a more effective and efficient means for scheduling and controlling network resources (Wright and Aitken, 1990:484).

The task of scheduling the network assets effectively is a challenging problem of supervisory control. For example, on any given day, interrelated information depicting nearly 1600 entries of satellite visibilities and scheduled network support must be interpreted and used to make decisions that are critical to the survival of on-orbit resources (Wright and Aitken, 1990:484). The AFSCN scheduling system must be flexible enough to react to and resolve problems associated with changing mission priorities, network equipment outages, and satellite anomalies.

As discussed previously, early network scheduling was performed using a paper contact acquisition chart. The horizontal axis of the chart represented time, and the vertical axis showed each ground station of the AFSCN (RTS). A single paper chart denoting a 24-hour period measured 36 inches vertically and 144 inches horizontally, discernible to one-minute increments. Three types of schedules were maintained concurrently: a seven-day forecast, a 24-hour schedule, and a real-time schedule. Appendix A provides a more detailed discussion of AFSCN scheduling activities.

Basic Scheduling (BASCH)

In the late 1970's, Headquarters U.S. Air Force concerns over the potential single-point failure within the Air Force satellite command and control system led to the development and construction of the Consolidated Space Operations Center (CSOC) at Falcon AFB. CSOC would provide redundant, interoperable satellite command and control capability to augment the existing control center at Sunnyvale Air Force Station (now Onizuka AFB). In order to support the concept of interoperability, identical capabilities would exist at each facility and either facility would be able to conduct normal command and control operations.

As part of this effort, the CSOC System Program Office (SPO), Headquarters Space Division, Los Angeles Air Force Base, was tasked with the responsibility to develop a dual-node scheduling system (BASCH) to

replace the existing scheduling database system, SCRABL II. The new system would provide the capability to schedule satellite contacts simultaneously at the two geographically separated facilities. By 1983, a final system-level specification was produced. The system specification was based on capabilities of SCRABL II and the new requirements to support dual-node operations, as well as the need to provide connectivity between the satellite MCCs and the local RCCs. This connectivity would provide the MCCs with access to the scheduling database to support the generation of contact requests and allow viewing of the official AFSCN schedule for real-time conflict resolution.

BASCH was developed in accordance with this specification as a single computer program configuration item (CPCI) under the CSOC SPO Space Operations Center/Network Control Segment (SOC/NCS) option on the Data System Modernization (DSM) contract. DSM was an ongoing Air Force program and contract to procure a new command and control segment (CCS) for the AFSCN, including modified facilities, commercial computer hardware and software, and new application software. Originally, the SOC/NCS option was to provide the CCS equipment for CSOC control centers, but in 1984 was modified to develop the BASCH scheduling software for the AFSCN as well. To provide commonality with existing systems, the developer recommended BASCH be designed to operate in the CCS software environment. The developer's implementation approach dictated the interdependence between the BASCH system and other CCS

CPCIs (e.g., CPCI-201 - Operations Planning, CPCI-205 - MCC Management, CPCI-209 - Common Services, CPCI-210 - Display Management, and CPCI-232 - External Interfaces).

The developer's design focused on producing software necessary to support scheduling activities on CCS mainframe computer systems in the two geographically separated locations. The developer's operational concept called for the two systems to run concurrently using the same database and to exchange updates to the database via a real-time communication interface.

The BASCH development effort proceeded as planned with the software Critical Design Review occurring in August 1985, followed by coding and unit testing. The next major milestone in the project was Functional Qualification Test (FQT). Originally scheduled for November 1986, FQT was conducted in May 1987 at the developer's facility in Gaithersburg, Maryland. The FQT provided a comprehensive test to validate the capabilities of the BASCH system and to verify that all BASCH B-level specification requirements, except those which could only be verified during operational testing, were incorporated into the system. The FQT plan and procedures identified 725 testable requirements. Upon completion of testing, 94 percent of the requirements had been verified.

Following FQT, the BASCH system successfully passed Functional/Physical Configuration Audits and proceeded to component testing. Testing of the major BASCH components was successfully

completed in December 1987. The next milestone was system testing which began in early 1988 but was never successfully completed. Initial system testing failed due to numerous problems discovered with internode data transfer between the two mainframe computers and overall system performance.

Originally scheduled for operational activation in October 1987, the availability of the BASCH system was delayed due to software problems identified during system testing and reprioritization of development resources. Over the next three years, the developer continued the effort with little progress made toward operational activation.

From 1987 to 1990, several operational exercises of the BASCH system were conducted by the CSOC program office site activation task force (SATAF). The results of one exercise noted that: 1) numerous critical problem areas need to be fixed before scheduling operations could transition to BASCH; and 2) the BASCH system lacks the robustness and stability to sustain operations. The report also stated " the negative aspects [of BASCH]... preclude this system from being used operationally, even when [operated] in a single node configuration" (SATAF/TE and The Aerospace Corporation, undated:3-4).

With the advent and operational activation of the ASTRO system in January 1991, all development and maintenance efforts associated with the BASCH project were stopped. In November 1991 after an assessment of the capabilities of BASCH and ASTRO, AFSPACECOM relieved the CSOC SPO

of responsibility for delivering the BASCH system and submitted the necessary requirements documents to implement a fully operational ASTRO system (Logemann, 1991:1).

Automated Scheduling Tool for Range Operations (ASTRO)

ASTRO is a computer-based scheduling system used to control and allocate ground-based resources in support of the AFSCN. The development effort was managed by the Satellite Control and Data Handling Program Office, Headquarters Space Systems Division, Los Angeles Air Force Base (SSD/CW). The ASTRO project was started as a proof-of-concept analysis to assess the potential availability of commercial hardware and contractor-developed software to support scheduling and allocating of AFSCN network resources, as well as eliminate the need for the paper acquisition chart. The development process associated with ASTRO can be broken into five distinct phases.

Phase I - October 1987 to December 1987. During the initial phase of the project, the developer focused on assessing existing AFSPACECOM range scheduling requirements, analyzing the content of the paper acquisition chart, and examining the functional interaction between scheduling personnel and the chart. In addition, the developer conducted an industry survey of available computer hardware, and investigated various alternatives to enhance the man-machine interface between the operator and the scheduling computer system (Unisys, 1987:2-4). The goals

of the initial analysis were to obtain a clearer understanding of the users' requirements and establish a basis for future development efforts.

To perform the analysis of range scheduling requirements, the developer reviewed the existing AFSCN Range Scheduling Operational Concept and the Network Control System/Statement of Operational Requirements Document . The developer also looked at the documentation produced during the BASCH effort (e.g., System Specifications, Test Reports, and System Deficiency Reports).

The analysis of the paper acquisition chart focused on the information and data presented on the chart which facilitates manual scheduling activities. The analysis provided a detailed understanding of the format of the chart, the graphic symbols and scheduler annotations associated with each scheduled event, the sources of scheduling data, and the use of each chart element (Unisys, 1987:4). The data gathered from this analysis was critical in establishing the system requirements for the display subsystem of ASTRO.

In analyzing the functional interaction between scheduling personnel and the chart, the developer examined existing scheduling procedures, observed current scheduling operations, and conducted in-depth discussions with senior scheduling personnel. It is important to note that three members of the ASTRO staff previously worked in scheduling with a combined 25 years of experience, so the analysis focused on gathering information beyond the current level of understanding among the staff.

As part of the man-machine interface effort, the developer investigated alternative data entry techniques (e.g., display selection devices and voice input/output products). The developer also reviewed a recent human factors study which focused on range scheduling operations and computer displays. The study reported that accessing multiple sequential displays forces excessive reliance on operator short-term memory, which results in increased error rates. In addition, the error rate increases with the number of additional accesses required and with the time required to perform those accesses. The study concluded that due to limitation of short-term memory, schedulers would be more efficient if they could see large amounts of data at one time as opposed to paging/scrolling through numerous computer displays and extracting the necessary data to make decisions (Unisys, 1987:9; Egbert and others, 1986:6).

In surveying existing computer hardware, the developer determined that the limiting factor for any future scheduling system would be the display subsystem. Thus, a large amount of time was allocated to assessing state-of-the-art display technology against the display requirements generated by the paper chart analysis.

The results of this initial analysis showed that current hardware technology could support the development of an improved AFSCN scheduling system; however, existing requirements documentation did not fully define all the processes associated with scheduling resources. At this point, SSD/CW had two viable development options: 1) produce the

necessary detailed system specifications and manage the effort under the traditional waterfall approach; or 2) conduct the project using an alternative development methodology to assist in the definition and clarification of the users' requirements. Past efforts (i.e., BASCH) had shown that it was extremely difficult to identify all the requirements for satellite scheduling and to translate these requirements into system-level specifications. Furthermore, because the availability of the system was critical to continued operations of the AFSCN and development of a detailed specification would be a time consuming task, the decision was made to allow the developer to construct the computer system using the alternative evolutionary prototyping approach.

Phase II - January 1988 to February 1989. Over the next thirteen months, the ASTRO project evolved from a conceptual design to a fully-operational prototype workstation system. During this phase, the developer used evolutionary prototyping to build a prototype system which contained most of the characteristics and the performance baseline of an operational system. The ASTRO prototype provided an interactive database to support the generation and manipulation of a large-screen display (22" x 34") to replicate the format of the paper acquisition chart. Additionally, the prototype contained a fully functional database and library of utility routines critical to scheduling activities.

The response to the prototype system by scheduling personnel was overwhelmingly positive, since they had played an integral role in the

development of both the requirements and the system design through periodic evaluations. By evaluating the prototype system, the users were able to provide feedback to the developer regarding the system's requirements. For example, during this period, the developer conducted over 20 system demonstrations for more than 120 individuals with space operations and scheduling backgrounds. These demonstrations provided feedback and suggestions for system improvements.

The resulting ASTRO prototype provided an operational system which fully replicated the paper acquisition chart and existing scheduling database. In addition, the ASTRO system provided several benefits beyond the existing paper chart and computer database. These benefits included the ability to share electronic representation of the range schedule with more than one facility, the elimination of errors due to the transcription of data from the chart to the computer database, and the automatic identification of schedule conflicts (Unisys, 1988b:18-20).

Phase III - February 1989 to March 1990. During the next phase of the project, the developer was tasked to expand the ASTRO system from a stand-alone workstation to a multi-user environment. The developer also continued improving and refining the existing ASTRO software based on feedback received from the schedulers. In addition, HQ AFSPACECOM requested that SSD/CW install the prototype workstation in each RCC for the purpose of conducting a formal users' evaluation.

To generate requirements for the multi-user enhancement to ASTRO, the developer conducted a further analysis of the functions accomplished by scheduling personnel at the paper chart. The analysis concluded that at any given time, up to eight schedulers could be working at the chart. Thus, an ASTRO system capable of supporting full operations must include a minimum of eight workstations. The developer also investigated two alternative system designs: centralized "time-share" processing and a distributed architecture (i.e., network) to support a multi-user environment. The analysis concluded that the most expedient means of providing multiple users access to the ASTRO database would be to network together several ASTRO systems using a Local Area Network (LAN) (Unisys, 1989:2-3). The decision to use a LAN was partially based on lessons learned from the BASCH effort, which used a centralized processor to control scheduling data.

Based on the decision to use a LAN, the developer examined the operational activities of each scheduling position to determine the expected loading on the ASTRO network server and central database system. The analysis results were used to derive requirements to evaluate commercially-available network products and to drive refinements to the custom ASTRO database management software.

From August 1989 to December 1989, the ASTRO prototype underwent extensive evaluation by personnel at both RCCs. The prototype was evaluated in a single user environment to assess the feasibility of the

ASTRO system replacing the acquisition chart. The evaluation rated the prototype in four areas: functional requirements, system performance, operator interface, and overall system capabilities. Each requirement within these four areas was measured based on a general evaluation and five specific criteria:

> 1) Requirements Satisfaction--how well did the prototype meet the specific requirements?
>
> 2) Scheduler Friendliness--how well does the system support the abilities and needs of the scheduler?
>
> 3) Reliability--scheduler's assessment of the system's performance during specific requirement evaluation.
>
> 4) Human Factors--evaluator's evaluation of the interaction with the system.
>
> 5) Timeliness--evaluator's assessment of the system responsiveness during the specific evaluation. (Detachment 9, 1990:2-3)

Each requirement was evaluated based on both a numeric rating and a text description. The numeric rating ranged from "1" to "5", with "1" signifying the lowest or poorest value assignable, and "5" signifying the requirement was evaluated as excellent or exceeded the evaluator's expectation (Detachment 9, 1990:3).

The prototype ASTRO system attained an overall rating of 3.73 out of 5 at Falcon AFB and an overall rating of 4.65 out of 5 at Onizuka AFB. The report suggested the difference in the rating could be attributed to the Onizuka AFB evaluators' familiarity with the ASTRO system due to the

close proximity of the ASTRO development facility, and the hardware problems encountered by Falcon AFB evaluators (Detachment 9, 1990:3).

The final evaluation report concluded that "the ASTRO system proved the feasibility of hosting scheduling data in a graphical format on an electronic medium" (Detachment 9, 1990:7). Additionally, the reports suggested that the evaluation provided a useful tool in providing operator feedback to the developer.

Phase IV - April 1990 to January 1991. Based on the success of the users' evaluation of the ASTRO system, the RCCs submitted formal requirements to Headquarters (HQ) AFSPACECOM requesting the activation of the prototype ASTRO system to support interim operational network scheduling. The RCCs requested the installation of a mini-ASTRO network (two ASTRO workstations and a database file-server) in each RCC. The ASTRO systems would work in conjunction with the existing paper acquisition chart to provide a limited dual-node scheduling capability. Each RCC would maintain the acquisition chart with one facility having primary scheduling responsibility while the other would remain a back-up. The databases of each ASTRO file-server would be updated periodically via a dial-up 9600 BAUD secure telephone modem. The operational configuration was considered an interim capability until some time in the future when the full ASTRO network, consisting of eight workstations per RCC, could be prioritized within existing budgets.

SSD/CW received the formal requirements for the interim ASTRO system from HQ AFSPACECOM in May 1990. After validation by the AFSCN operational community, SSD/CW tasked the ASTRO developer to procure the necessary hardware, produce the required documentation to support the activation, and conduct formal development testing. In September 1990, interim ASTRO systems were installed in both RCCs. Throughout the development of the interim ASTRO system, the software underwent a continuous improvement and enhancement process in which requirements were internally prioritized based on their impact to the pending activation.

In October 1990, the two ASTRO systems were turned over to AFSPACECOM for operational test. Scheduling personnel and the Air Force Operational Test and Evaluation Center conducted a three-month operational test of the ASTRO system. Problems found during these tests were communicated back to the developer, who incorporated the corrections into a new software delivery. In January 1991, HQ AFSPACECOM decided to formally activate the interim ASTRO system to support operational scheduling. At the time of activation, only two minor software problem remained open against the operational software baseline. Finally, with the activation of the ASTRO system, the SCRABL II computer system was deactivated after 22 years of service supporting AFSCN range scheduling.

Phase V - January 1991 to July 1992. The ASTRO system activated in January 1991 provided only an interim scheduling capability until such

time as an alternative could be identified and procured to satisfy near-term operational scheduling requirements. From May to August, 1991, SSD/CW assessed near-term requirements against the two system alternatives, BASCH and ASTRO. The assessment evaluated each alternative against existing operational requirements, and also considered the timeframe in which either system could provide the necessary capabilities. The assessment concluded the ASTRO system was the only alternative which met the users' requirements and could be delivered in the required timeframe. In a 12 November 1991 letter to SSD/CW, the HQ AFSPACECOM Director for Force Support stated:

> We have reviewed the results of the Near-Term Requirements Implementation Assessment, and considered your recommendation that we jointly pursue acquisition of an enhanced ASTRO system to meet our near-term range scheduling requirements. We fully agree that this solution, further developing the ASTRO system hardware and software and establishing an internode communications link between the two RCCs, is the most feasible and lowest risk option available to meet our needs and timeline requirements. (Logemann, 1991:1)

Based on the above letter, SSD/CW tasked the ASTRO developer, via an Engineering Change Proposal (ECP), to produce and install an ASTRO system to meet near-term scheduling requirements. The tasking directed the developer to install eight ASTRO workstations, a database file-server and the associated system peripherals in each RCC. Additionally, the tasking required the installation of a T-1 communication link between the two sites. The operational concept for the ASTRO system called for all 16

workstations to be connected to a single database file-server with the second server periodically updated via the communication link. In the event the prime file-server failed, the secondary server could assume the prime role.

As of 31 July 1992, the ASTRO developer had completed hardware installation and initial testing and the system was formally turned over to AFSPACECOM for final testing prior to operational activation. In initial operational testing, several minor software deficiencies were identified. The ASTRO developer has already resolved most of these problems and has prepared a new software master for to delivery to the users (List, 1992).

V. Analysis and Findings

Now that the case background has been established, the second task of the case analysis is to address the investigative questions directly and present the findings. In addressing these questions, three general sources of data are used. The first source is documentation, consisting of range scheduling automation study progress reports prepared by the ASTRO developer, assessments by independent organizations and associate contractors, and meeting minutes. The second data source is archival records such as interoffice memoranda and letters. Finally, the third source is notes from unstructured interviews conducted with key participants in either the BASCH or ASTRO development, or both. Appendix D contains a brief summary of the experience and specific involvement of each individual interviewed. All of the data sources used in this analysis are included in the bibliography and are cited where appropriate.

Again, the overall research question is: Do the results observed in the BASCH and ASTRO projects support the claimed advantages and disadvantages of the evolutionary prototyping methodology as compared to the traditional waterfall methodology found in the literature? The investigative questions are:

1) What is the problem domain? Were the high level system requirements and objectives the same for both efforts?

2) What development methodology was used in the BASCH effort? Why was the methodology chosen?

3) What are the factors cited for the failure of BASCH?

4) What development methodology was used in the ASTRO effort? Why was the methodology chosen?

5) What are the factors cited for the success of ASTRO?

6) What theoretical advantages and disadvantages of the evolutionary prototyping were evident in the ASTRO development effort?

7) What other factors might have influenced the contrasting outcomes of the two efforts?

The following sections present a narrative analysis and findings for each of the investigative questions.

Investigative Question 1

What is the problem domain? Were the high level system requirements and objectives the same for both efforts?

The first question establishes the environment in which the BASCH and ASTRO systems were to be developed and operated and the objectives of the range scheduling automation endeavor. The advent of dual-node control operations within the AFSCN necessitated the development of a scheduling capability which could support operations from the two geographically separated locations. The concept required scheduling operations be performed concurrently at both locations or independently at either location should the other facility become inoperable or unavailable.

Both the BASCH and ASTRO systems were developed with these capabilities as the overall objective.

For the BASCH system, the overall system-level requirements were as follows:

> Provide an interoperable dual-node scheduling system with enhanced survivability, improved man-machine interface, and additional automation. The improved system should be able to handle an increased workload, and allow scheduling to be accomplished with a smaller, less-skilled (and hence less expensive) staff. (SDC, 1987:1)

Similar to the BASCH requirements, the ASTRO project was to provide a system to handle the increasing AFSCN workload and to provide an interoperable scheduling capability. The primary objective was to eliminate the need to conduct scheduling via the labor-intensive paper acquisition chart and transition scheduling operations to an interactive computer-driven display system. This display system would allow scheduling personnel to manipulate and maintain an "electronic" schedule at both facilities without the manual construction of duplicate charts. Furthermore, since the computer database, and/or changes to it, could be electronically transmitted, both facilities could conduct operational scheduling (Unisys, 1987:1-2).

For both systems, the lower-level requirements were very similar. For BASCH, the Air Force development team and the users defined top-level requirements in the A-level specification. These requirements directly influenced the development of the lower-level requirements in the BASCH

B-level specifications and eventually impacted the specific implementation approach. For ASTRO, the Air Force provided only minimal top-level direction and allowed the developer to establish lower-level requirements through the application of evolutionary prototyping methodology.

In summary, the findings suggest there were only minimal differences in the problem domain across the two development efforts. The top-level requirements for both systems were highly comparable; however, the most obvious difference between the two efforts was the methodology used to establish the lower-level requirements. For BASCH, the requirements were rigidly defined based on the A-level specification prior to program start. This approach, in keeping with the philosophy of the traditional approach to software system development, demanded the process be driven by increasingly more detailed levels of specification. In contrast, the ASTRO system benefitted from the derivation of lower-level requirements through the application of evolutionary prototyping, which provided the continuous definition and clarification of requirements throughout the development process.

Investigative Question 2

What development methodology was used in the BASCH effort? Why was the methodology chosen?

Although the DSM contract and the SOC/NCS option did not specifically require compliance with DOD-STD-2167, IBM developed BASCH

using the traditional waterfall model and deliverables such as formal documentation and reviews, and testing of the software. There was no real option at the time other than to use the traditional waterfall method of software development, as defined by DOD-STD-2167. Alternative methods were in their infancy and the DOD did not encourage their use. The traditional method was a proven, disciplined, systems engineering approach to computer system development. Although it was a separate option, BASCH was actually produced in the same software development environment as the CCS CPCIs. All the software was to be hosted on the large new DSM centralized computer architecture already designed and installed at Onizuka AFB and Falcon AFB.

Investigative Question 3

What are the factors cited for the failure of BASCH?

The third question addresses those factors evident in the data which played a part in the failure of the BASCH effort. The findings establish validity for these factors by using the three sources of data discussed in Chapter III and the introduction to this chapter. The data analysis identified eight factors which impacted the outcome of the BASCH effort. These factors are identified in Table 4 and will be discussed at length during this analysis.

The main factor in the failure of BASCH cited in the data is that, as specified, the BASCH system failed to meet all requirements needed to

Table 4.

Factors Influencing the Failure of BASCH

1.	The BASCH system requirements did not meet the users' requirements.
2.	Development approach did not provide a means to incorporate changing or new requirements.
3.	Lack of ownership among the users since they did not participate in the requirement process.
4.	Developer's proposal recommended the BASCH system operate in the CCS environment.
5.	Poor man-machine interface and system performance.
6.	Geographical separation between the developer and the users.
7.	Recurring problems with maintaining operator proficiency on the system.
8.	Low prioritization of BASCH maintenance action as compared to other CCS functional areas.

conduct AFSCN range scheduling activities. As noted, the BASCH project began in 1983. However, as early as 1985, the AFSCN Range Scheduling Working Group identified numerous functional capabilities critical to operational scheduling which were either inadequately defined in the BASCH specification or totally absent from the system. Two subsequent independent assessments confirmed the working group's findings.

In 1987, System Development Corporation (SDC) was tasked to conduct an assessment of the BASCH functional capabilities. The

assessment analyzed the operational capabilities of BASCH and reported there were 16 areas of major concern. The assessment reported "each of these problem areas had the potential to cripple operational use of the system" (SDC, 1987:6). Finally, the assessment concluded:

> the collective effect of all problems makes successful operations highly unlikely. It must be concluded that the baseline CPCI 350 [BASCH] system will not be capable of supporting operational scheduling. (SDC, 1987:6)

In July 1990, The Aerospace Corporation was tasked to conduct a second analysis of the functional capabilities of BASCH against operational scheduling requirements. Since in the time-frame of the analysis there did not exist a validated set of operational requirements, The Aerospace Corporation developed operational requirements based on the existing capabilities of the SCRABL II system. The assessment separated scheduling capabilities into 28 distinct functional areas. The analysis assessed each area against the capabilities of BASCH. Each area was graded based on "the operational usability" of the system capabilities. The criteria for each grade is shown in Table 5.

For the BASCH system, the report gave the following grades:

Meets Expectation	10 areas
Below Expectation	13 areas
Not Supported	4 areas
Not Required	1 area

(Miller and Springsteen, 1990,2-10-2-12)

Furthermore, the report recommended discontinuing further development of the BASCH system since it did not meet the users' requirements (Miller and

Table 5.

The Aerospace Corporation Functional Grading Criteria
(Miller and Springsteen, 1990:2-9)

Grade	Criteria
Exceeds expectation	Functionally exceeds SCRABL II.
Meets expectation	Functionally meets SCRABL II.
Below expectation	Functionally does not work properly under nominal conditions, a work-around is required, or functionality is extremely cumbersome.
Not supported	Functionality is not currently supported.
Not required	Functionality is not required since it is provided in a different function.

Springsteen, 1990:3-1).

Finally, it was noted by one individual who participated in the BASCH system specification review that the system specification was developed to meet cost as opposed to providing all the capabilities of the SCRABL II system. This individual added that during the Air Force system specification reviews, numerous users' requirements were disregarded by the program office, since their addition to the specifications would drive the cost beyond the available budget (List, 1992).

The second factor is the inability of the development process to incorporate changing or new requirements into the system. As discussed previously, numerous requirement changes were identified during the

BASCH project; however, since these requirements were beyond the scope of the original specifications, they were deferred until completion of the initial development. As the time between the development of the specifications and the date of availability increased, so did the number of requirement changes and the cost of incorporating these changes into the system. This growth in development costs supports the theory that it is significantly less expensive to fix requirements problems early in the effort as opposed to the maintenance/support phase.

The third factor relates to the impression among scheduling personnel that the BASCH system did not reflect their operational requirements. It was noted that as a result of being ignored during the requirements phase, the schedulers felt no "ownership" of the BASCH system (List, 1992; Wong, 1992). In addition, during initial tests of BASCH in the RCCs, the schedulers identified and documented numerous software problems. The BASCH developer closed many of these problems as "operator error" when in fact they were valid problems (List, 1991:3). These actions led to a sense of distrust between the users and the program office and the BASCH developer which resulted in a general unwillingness among the schedulers to accept the system until it met all of their requirements.

The fourth factor involves the environment in which the system was developed. As discussed in Chapter IV, the developer proposed to design BASCH to operate in the existing CCS environment. The developer's intent was to take full advantage of existing CCS functional capabilities (e.g.,

Display Management, MCC Management, and Common Services), which should in theory have reduced the overall cost and risk of the development. However, what the developer produced was a scheduling "transaction-based" system to operate in an environment designed to support real-time satellite command and control (Simpson, 1992). Thus, BASCH suffered from not only its own problems but also the problems of the entire CCS system.

The fifth factor relates to an extremely poor man-machine interface in BASCH. Early in the development, it became obvious to the operational scheduling community that the man-machine interface would make or break BASCH. The AFSCN Range Scheduling Working Group discussed at great length the capabilities of BASCH in regards to this area. Numerous requirement changes were identified and included in the list of future BASCH modifications to support area improvements. In addition, the 1990 Aerospace Corporation assessment stated, "the man-machine interface for many BASCH functions is awkward to use" (Miller and Springsteen, 1990:2-7). The report went on to identify six specific functions in BASCH with awkward man-machine interface.

A further example of the poor man-machine interface of BASCH was the function used by MCCs to generate contact requests. In the CCS system, MCC planners were to be provided with two methods to generate satellite contact requests, CPCI-201 (Operations Planning) or BASCH. However, the request entry frames for CPCI-201 bore no resemblance to the

request entry frames of BASCH; thus, MCC planners were required to learn two different formats (List, 1991:1).

Another man-machine interface problem was the number of screen accesses required to obtain necessary scheduling information for conflict resolution. The human factors study by Egbert and others stated:

> A timing analysis [of BASCH] suggests that this process will be significantly slower than the current method, due in part to the display update, and in part to the reliance on human short term memory... The multiple screen requirement introduces the risk for errors in data transcription, provides less information per display, less flexible information manipulation and risks potential non-acceptance by operators. (Egbert and others, 1986:6)

The analysis concluded that the introduction of a poorly engineered automation could actually increase scheduler loading through increased "cognitive demands." These demands could in turn have a detrimental effect on the abilities of schedulers to handle increased AFSCN contacts (Egbert and others, 1986:18-19).

The sixth factor involves the geographical separation between the system developer and the users. BASCH was developed at IBM's facility in Gaithersburg, Maryland, while the schedulers were located in California and Colorado. Also, the exchange of data between the users and developer was limited to formal design and documentation reviews. In addition, all questions were channelled through the developer's management for presentation to the users--there was no direct interface between the BASCH analysts and programmers and the users. Thus, the analysts and

programmers had little insight into the problem domain for which they were developing the system and the physical separation further supported the feelings of alienation between the developer and schedulers (List, 1992; Simpson, 1992).

The seventh factor refers to the recurring problem of maintaining operator proficiency on BASCH. To support BASCH activation, scheduling personnel were required to maintain system proficiency to support system and initial operational testing. Prior to the original activation date, 1 October 1987, scheduling personnel underwent extensive BASCH training. This training familiarized schedulers with the functions and capabilities of BASCH. Nevertheless, with the continual slips in system availability, the proficiency of the users diminished over time. During this period, the schedulers were asked to conduct operational scheduling, which required proficiency on SCRABL II or ASTRO, while also trying to maintain BASCH proficiency by exercising the limited capabilities of the existing BASCH system.

A 27 April 1990 internal CSOC SPO memorandum implied the pending ASTRO interim scheduling system would directly impact the current BASCH internode system test schedule. The memo stated:

> IBM reports the system test would move out one day for each day ASTRO is used for full interim operations. This is due primarily to one factor: manpower. At current manning levels, full-up ASTRO operations in the RCCs would require all human resources assigned. [The BASCH] system requires the involvement of BASCH-proficient operators. (Pope, 1990:1)

The memo also recommended scheduling personnel at Falcon AFB conduct 7-day scheduling operations on BASCH, while maintaining proficiency on the ASTRO system. ASTRO proficiency was required to support the back-up of scheduling operations at Onizuka AFB by Falcon AFB schedulers (Pope, 1990:1-2).

The final factor cited in the research is the impact of the low prioritization BASCH maintenance activities were given in comparison to other CCS functional areas. In October 1988, the SOC/NCS option to the DSM contract was contractually closed out. Since all BASCH software deficiency reports (S/DRs) had not been resolved, corrective action was transferred to the Command and Control Sustaining Engineering (CCSE) contract. The CCSE contract was a follow-on sole source award to IBM to provide level-of-effort resources for CCS software maintenance and ECP system improvements.

Under the CCSE contract, the developer was directed to provide CCS model deliveries every six months. Each model delivery would include approximately 500 individual CCS S/DR corrections and was considered a stand-alone software delivery. Upon successful system test, the model would become the baseline CCS software and would be installed in all AFSCN control centers. The content of each model delivery was based on a prioritized list of the CCS functional areas, as well as the available CCSE maintenance resources.

It is important to note that in the model planning process, BASCH was prioritized seventh out of eight CCS functional areas. Thus, fixes to BASCH were extremely slow to be delivered. For example, from October 1988 to November 1989, 24 total BASCH S/DRs out of approximately 130 outstanding S/DRs were included in five model deliveries (Models X, Y, Z, 1, and 2). BASCH did not fair any better in the next three models. Only four S/DRs were delivered in Model 3 in April 1990, ten S/DRs in Model 4 in November 1990, and 17 S/DRs in Model 5 in April 1991. Further impacting BASCH progress was the fact that with each subsequent model delivery, a new set of BASCH problems were identified (Simpson, 1992). These new problems would then flow into the CCS model planning system for resolution in some future model.

The July 1990 Aerospace Corporation assessment identified 145 outstanding BASCH-related S/DRs which were to date unresolved. Of these 145 S/DRs, 31 S/DRs were scheduled for delivery in an upcoming model, 99 S/DRs had not been scheduled for a model delivery, and 15 where identified as enhancements to BASCH, which put them outside CCS maintenance responsibilities (Miller and Springsteen, 1990:2-6). The assessment suggested the development effort required to resolve these critical problems, which limited the use of BASCH, would be substantial. For this reason, BASCH would not be ready to fully support AFSCN scheduling operations after the delivery of CCS Model 5 or even within "a reasonable period of time afterward" (Miller and Springsteen, 1990:3-1).

Further evidence of the problems associated with BASCH maintenance was provided in a 18 September 1990 letter from the RCC Commander at Falcon AFB to 2nd Space Wing/DOQ. The letter identified nine flight-critical and 27 flight-limiting BASCH S/DRs which had not been scheduled for model delivery. The letter stated:

> The RCCs... have evaluated all outstanding S/DRs that have not been assigned to a model drop [delivery]. The deficiencies that are scheduled for delivery by Model 5 will not provide range scheduling with sufficient capability to support dual-node range scheduling with BASCH.
>
> In order to support range scheduling operations with BASCH, all BASCH fixes currently scheduled for Models 3, 4, and 5 must be delivered and function correctly. In addition, as a minimum, fixes identified in attachment 1 [nine flight-critical and 27 flight-limiting BASCH S/DRs] must be delivered. (Fruland, 1990:1)

In summary, the analysis of the data identified eight factors which impacted the outcome of the BASCH effort. It is impossible to determine if any one of these factors singularly could have led to the failure of BASCH; nevertheless, the analysis did reveal that all of these factors contributed to the failure of the BASCH development effort.

Investigative Question 4

What development methodology was used in the ASTRO effort? Why was the methodology chosen?

When ASTRO was undertaken, exclusive use of the traditional waterfall model, as mandated by DOD-STD-2167, had come under question and was being relaxed in the standard's revision. The AFSCN had its own

bad experience and was struggling not only with the BASCH effort but with the entire CCS system. At the start of the ASTRO project, the Air Force was considering all alternatives to make the new system operational.

The ASTRO development effort was conducted using several one-year duration study task orders on the Computer Program Integration Contract (CPIC), a level-of-effort contract managed by SSD/CW. The developer applied evolutionary prototyping to the entire effort, initially calling it "an experiment in system prototyping" (Unisys, 1988a:8). The "incremental" nature of the tasking should not be confused with incremental software development described in Chapter II. The study was not initially conducted to deliver an operational system; rather, it evolved into a system development effort when other alternatives were determined inferior.

The first task order requested the developer to conduct a study consisting of two primary phases, with the objective "to determine the feasibility of replacing the Range Scheduling Acquisition Chart with large-screen computer-driven interactive displays." During the first phase, the developer conducted a requirements analysis and equipment selection for a prototype system. Phase 1 concluded with a report which detailed the study steps, the method and results for derivation of display requirements, results of vendor surveys for equipment, recommendations for equipment lease/purchase, and an overview of the plans for the next phase. The report also included the first detailed description of the chart format and contents (Unisys, 1987:2-27).

During the second phase, the developer finished procuring the equipment recommended in Phase 1 and put together the first ASTRO prototype, including about 20,000 lines of code, a significant portion of which was reused. The developer's progress report noted:

> Wherever possible, scheduling software designs were "ported" from the existing SCRABL II system. Many required functions were added to the ASTRO prototype by simply re-coding them from assembly language to Pascal. Many other routines were found applicable to the ASTRO system with minor re-design. Overall, there was minimum wasted effort to "re-invent the wheel." (Unisys, 1988a:6)

Because of this aspect, the developer considered the ASTRO prototype to be an evolution of SCRABL II. Phase 2 was extended to the end of fiscal year 1988 to continue evolutionary development of the prototype.

The reasons for selecting the evolutionary prototyping approach for ASTRO are provided in the Unisys report submitted at the conclusion of the second task.

> From the user's point of view, prototyping can minimize the paperwork overhead necessary to define, document, and maintain system requirements and specifications. Prototyping also allows a "fly before buy" evaluation, curing the common "that's not what I asked for" problem. From the developer's point of view, the absence of rigid specifications allows more creativity, provides the opportunity to explore more approaches to the system design, and allows very rapid development. (Unisys, 1988a:20)

The developer further suggested that the prototyping method could reduce documentation costs, by eliminating unneeded documents, and thereby increase productivity. "Constant feedback, rather than paperwork, is the key to a successful prototyping development" (Unisys, 1988a:47).

Investigative Question 5

What are the factors cited for the success of ASTRO?

The next question identifies factors cited in the data which influenced the successful outcome of the ASTRO project. Table 6 identifies those factors evident from the research.

For ASTRO, the developer's decision to apply the principles of evolutionary prototyping played an integral part in the project's success. Evolutionary prototyping provided: 1) a method to continually define and clarify users' requirements; 2) a communications loop between the users and developer; 3) a means to effectively respond to users' concerns and requirements; and 4) the flexibility in the selection of system design characteristics, as well as the hardware platform. Thus, the first four factors identified in Table 6 can all be attributed to the use of evolutionary prototyping throughout the development effort.

The literature review notes that in order to effectively produce a prototype computer system, the developer must first gain a clear understanding of the users' requirements. Therefore, as part of the evolutionary prototyping process, the developer must continually gather data on the users' requirements and assess them against the prototype's capabilities. Thus, the process of requirements definition and clarification continues throughout the development.

The first factor cited in the data impacting the outcome of ASTRO is the ability of the developer to acquire a clear understanding of the users'

Table 6.

Factors Influencing the Success of ASTRO

1.	Strong understanding of users' requirements and needs by the developer.
2.	Continuous flow of feedback between developer and users regarding system capabilities.
3.	Developer was able to quickly respond to new users' requirements and software errors.
4.	Developer given flexibility in system design and hardware architecture.
5.	Related experience of the development personnel.
6.	Lessons learned from past scheduling development efforts.
7.	Strong commitment to the development process and end product--developer initiative.

requirements. As discussed in the case background, initial ASTRO efforts focused on a detailed analysis of users' requirements. The analysis was not limited to written documentation, but also included the observation of actual scheduling operations. Furthermore, the developer did not limit the time-frame of the requirements analysis to just the preliminary phases of the project. Instead, the developer continually reexamined the users' needs against the prototype system. Thus, the system capabilities kept pace with changing users' requirements.

The users also clearly understood the benefits of using prototyping to assist in the definition and clarification of system requirements. A 1989

letter from the Director of Space System Planning, HQ AFSPACECOM, to SSD/CW stated the primary purpose of the ASTRO effort "is to determine if technology is capable of meeting long-term scheduling requirements and to aid in developing system specifications" (Pagano, 1989:1-2). The letter added that one of the goals of ASTRO was to establish users' requirements by enlisting and documenting specific users inputs and establishing detailed range scheduling requirements (Pagano, 1989:2).

SSD/CW management also understood the benefits prototyping provided for the definition of requirements. The Program Director wrote:

> The AFSCN processes on the order of a thousand separate requirements each year to enhance network capability to meet the support needs of various satellites. Many of these individual requirements are translated into modifications or additions to the network assets. Prototyping has been utilized successfully for complex and urgent developments to meet many of these requirements. Rapid prototyping has also been used for requirements definition and for defining man-machine interfaces. Through prototyping, the AFSCN has... improved the process of defining requirements for operational satellite support systems. (Whipple and Hoida, 1991:1)

The second factor delineates the impact users' feedback had on the ASTRO outcome. Each interviewee identified the close working relationship between the developer and users as a primary reason for the success of ASTRO. A range scheduling manager noted that the tight users/developer loop fostered cooperation, which benefitted both groups. The users were getting a system which met their needs and the developer was getting continual constructive feedback on the capabilities of the prototype system (Wong, 1992).

The third factor relates to the timely response evident throughout the ASTRO development effort. An examination of the software deficiency/ enhancement process used during the effort provided a clear picture of the developer's responsiveness. With the installation of the ASTRO single-user prototype in the RCCs for formal evaluation in August 1989, the ASTRO system software was placed under informal configuration control. After August 1989, baseline changes were tracked via ASTRO Report Forms (ARFs). The ARFs provided an audit trail of software deficiencies during the prototype effort, as well as a means for AFSCN scheduling personnel to document suggested improvements and/or enhancements to the system (Miller and Springsteen, 1990:2-3). ARFs were generated by the development team, scheduling personnel, and test personnel. Each ARF was logged against the current system by the developer and tracked until it was closed by a future software delivery. Prior to the ARF process, system deficiencies and suggested enhancements were informally tracked by the developer.

The success of the process can be shown by analyzing the number of ARFs processed and the number of new masters delivered. From August 1989 to January 1991, nearly 800 ARFs were generated against the ASTRO system. Upon operational activation in January 1991, 94 percent of the ARFs had been incorporated into the system baseline and 34 ASTRO software masters had been delivered. Of the remaining open ARFs, only

two were considered software deficiencies and both were considered "noncritical" to operations by the schedulers.

From January 1991 to October 1991, an additional 225 ARFs were generated. During this period, the developer delivered six new software masters and four expedited deliveries, closing a total of 212 ARFs and resolving all identified software deficiencies.

The ASTRO developer also benefitted from the fact their development facility was in close proximity to the scheduling personnel located at Onizuka AFB, Sunnyvale, California. The developer's proximity facilitated the timely response to users' concerns, since the developer could obtain immediate feedback regarding system design changes, software deficiency corrections, or new functional capabilities.

The fourth factor describes the flexibility provided to the developer in the design and development of the system software and the evolution of the hardware architecture. The ASTRO system benefitted from the developer's ability to experiment with various design alternatives and evaluate each alternative through the feedback process. This flexibility allowed the developer to use a trial and error approach without the threat of retribution from the Air Force in the event of failure (List, 1992). In contrast, one interviewee noted that the BASCH developer was severely penalized via an award fee reduction for taking the initiative in resolving BASCH-related deficiencies, since their actions were not in keeping with the direction of the CCS functional priorities (Walker, 1992).

For ASTRO, the contract tasking provided the developer with the flexibility to improve the system hardware in conjunction with advances in computer technology. For example, initial development efforts were conducted on 386/20 personal computers (PCs), but with advances in technology, by 1992 the ASTRO system operated on 486/33 and 486/50 PCs. In addition, based on users' concerns dating back to the August 1989 evaluation, the developer continually evaluated alternative display systems. In 1992, the developer replaced the initial Greyhawk large-screen displays, which were based on a rear-projection laser system with state-of-the-art Sony (2K x 2K) CRT displays. In contrast, the BASCH developer was locked into the hardware architecture of the CCS system, which consisted of early 1980's mainframe computers. Thus, the ASTRO system was able to take advantage of these advances in computer technology, which resulted in improved system performance.

The fifth factor relates to the experience of the developer's staff. The developer clearly understood the problem domain and operational process in which the ASTRO system was intended to operate. This understanding was based on a combined 25 years of operational AFSCN scheduling experience, as well as a combined 80 years SCRABL II system software maintenance experience of the ASTRO staff. This experience resulted in a strong working relationship between the schedulers and the developer, since the ASTRO staff could empathize with the problems associated with range scheduling (List, 1992). In contrast, an Aerospace system engineer noted

numerous BASCH problems could be attributed to a lack of AFSCN scheduling experience by the development staff (Simpson, 1992).

The sixth factor describes the importance of lessons learned from past development efforts and how these lessons were applied in the early phases of ASTRO. It was noted in discussions with the ASTRO project manager and an IBM systems engineer that the ASTRO developer benefitted from past scheduling development efforts, SCRABL II and BASCH, and the lessons learned from analyzing their successes and failures. The developer used numerous aspects of the SCRABL II system to directly influence initial ASTRO designs. As discussed earlier, approximately 20,000 SLOC of SCRABL II software were recoded into Pascal to support the ASTRO effort. Furthermore, the developer was able to analyze the problems of BASCH and assure they were not duplicated in the ASTRO project (List, 1992; Walker, 1992).

The final factor apparent in the research recounts the sense of commitment evident in the development staff. A range scheduling manager suggested the developer displayed a strong commitment toward the project and process, as well as the end product. He added that the ASTRO developer consistently took the initiative to make the system work, and this commitment and initiative further strengthened the working relationship associated with the project (Wong, 1992).

In summary, two things stand out in the analysis of the factors which made the ASTRO project so successful: first, the ability of the developer to

define a very complex set of requirements and translate those requirements into an operational system; and second, the cooperation between the developer and users and their ability to work toward a unified goal. These ideas are both associated with advantages of the selected development methodology, evolutionary prototyping. However, it would be incorrect to infer the development methodology alone determined the outcome of the ASTRO project. The analysis has explicitly shown there were numerous other factors which played a role in the success of ASTRO.

Investigative Question 6

What theoretical advantages and disadvantages of the evolutionary prototyping methodology were evident in the ASTRO development?

Eleven advantages of prototyping over traditional waterfall development were presented in Table 1, page 24. Each will be addressed with respect to the ASTRO development effort in the following paragraphs. Specific examples or supporting information from the data are incorporated to augment the findings.

First, prototyping provides a means for users and developer to work together defining and building a system. As stated earlier, this was clearly a significant contributor to ASTRO's success. "Forty demonstrations were given to 320 people with space operations and scheduling background, over a three year period, to solicit suggestions and to guide the development process" (Whipple and Hoida, 1991:3). The demonstrations, ARF process,

and scheduling working group meetings provided a quick means for users not only to report problems, but also to make suggestions for future system improvements. Prototyping allowed the users to influence the design and "see and feel the results of the development process 'hands-on' rather than write requirements documents or read design documents" (Unisys, 1988a:47).

Second, prototyping provides greater insight into the users' requirements than the traditional approach. The range automation study started with an in-depth analysis of the paper acquisition chart and the functions performed by schedulers using the chart. Although this seems like an obvious place to start in an automation attempt, a detailed description of the chart format and contents had never before been accomplished. The developer concluded that "the current scheduling procedures and chart annotation techniques cannot be lightly dismissed" (Unisys, 1987:7). As a result, "the ASTRO Network Display also duplicates important subtle features of the acquisition chart which have never been formally described in requirements specifications, but which are taken for granted by the scheduling staff and used in routine operations" (Unisys, 1988a:9).

An example of the difference in the requirements insight achieved through the two approaches is provided in Unisys's first study report:

> [The schedulers'] stated requirement is to view twelve hours of schedule and visibility data for all tracking stations (the entire network) on a single display. The chart emulation features of

> previous scheduling computer systems (the site plot displays of the SCRABL and [BASCH] systems) have been limited to very small sub-sets of the required data... Many observers have assumed that the schedulers' stated need for a display spanning twelve hours of schedule is based on experience with the static paper chart, and that panning, scrolling, zooming, windowing, and split-screen techniques might suffice to provide an equivalent capability. (Unisys, 1987:9)

The ASTRO developer never compromised on this requirement; the objective of the study remained to provide a large screen display which would span the full twelve-hour period.

Prototypes can be easily modified to accommodate changing users' requirements. During the three-month period of Phase 2 extension, seven hardware upgrades and 16 software enhancements were implemented on the ASTRO prototype. The following 4.5 month period produced 18 software enhancements. All of these changes were accomplished within the original number of hours estimated at the beginning of the period because the developer was able to respond quickly to changes and avoid going too far down the wrong path.

As a result of prototyping, the end product is a more stable, tangible representation of the users' requirements. The ASTRO prototype effort was considered complete on 13 February 1989, or at least the objectives of the original study had been met. ASTRO workstations were subsequently installed at the two operational locations for evaluation by the users. While the network display and database scheduling functions were "a superset of the capabilities of the operational scheduling system," the system could not

be considered stable (Unisys Report, 1989a:5). This is because the original prototyping effort was limited to the network display and scheduling functions. Because of its success, the ASTRO effort was expanded in scope to go beyond the original objectives and provide networked workstations, internodal operations, and hard-copy schedule output.

Prototyping provides a better basis upon which to estimate the time and effort required to develop the final system. In the second study report dated 30 June 1988, Unisys estimated the total software for the final ASTRO system would be 35,000 to 40,000 lines of code, and completion would occur on 31 December 1989. At the time they had coded or reused 20,504 lines of code, so this meant that approximately 15,000 to 20,000 lines of code would be added. In the 13 February 1989 report, when the system was deemed complete, the total software lines of code was 44,838, or an increase of 24,334. This 25 percent growth from anticipated size corresponds with the 25 percent increase in the time required to complete, so the software productivity level was right on target. Many new enhancements had been added which were unanticipated in June 1989, but it cannot be determined from the available data if this accounts for all of the variance.

In each of the study progress reports, the ASTRO developer provided estimates of the hardware costs for the next phase of the project. All hardware was commercially available, so estimates were easily obtained. ASTRO hardware costs remained within the conservative $100,000 per

workstation estimate provided in June 1988. Documentation costs were estimated in June 1988, based on 40,000 lines of code for the final system; however, documentation of ASTRO has not been completed.

The prototyping process provides early visibility and elimination of development problems. There were no specific instances of this advantage noted in the data or interviews. However, frequent meetings with the Air Force program manager and users provided significant visibility and minimized the chance of development problems occurring. One system engineer noted that these meetings with the schedulers prevented the developer from wasting time and resources on "excursions down the wrong path," and in so doing reduced the overall schedule (Simpson, 1992).

Development risk and uncertainty are reduced with prototyping. Although this advantage was realized in the ASTRO effort, Unisys initially cautioned that both the developer and Air Force were at significant risk in a prototyping effort because "should disputes or misunderstandings arise, neither party has firm specifications to settle the argument" (Unisys, 1988a.20). The risk reduction was achieved because the commitments of time and resources were relatively short (compared to BASCH) and the results were highly visible.

There was very little cost risk in the original ASTRO prototyping effort, since it was developed under incremental task orders and the process could have been stopped at any time if progress was not satisfactory. The labor hours and hardware cost estimates for the following phase were

provided at the conclusion of each phase and could be controlled through the task orders. Decisions to change or enhance the hardware or software had to either be made within the authorized expenditures and the allotted budget of hours, respectively, or requested in future phases.

Two potentially significant cost risks did occur in the ASTRO development. The first was the ill definition of the level of documentation which would be required for the system after it was determined that the system would become operational. The documentation issue was deferred until recently, and the costs are significant if a maintenance/support contract is to be awarded by competitive bid. The second cost risk was the possibility that the Air Force might require that the ASTRO software be recoded in the DOD standard Ada language. In July 1990, The Aerospace Corporation estimated that the requirement to recode to Ada would add $7.8 million to the development cost of ASTRO, or 2.5 times the life cycle cost of the system as of the report date. This cost did not materialize, but the determination was not made until after the ASTRO system had been fully developed.

Although it was not intended at the outset to become an operational system, ASTRO later turned out to be the least risky system from a technical and schedule aspect as well, when compared to BASCH. In a 12 November 1991 letter, the using organization stated that continuing ASTRO development and discontinuing all BASCH effort was "the most feasible and lowest risk option available to meet our needs and timeline

requirements" (Logemann, 1991:1). It was emphasized by one systems engineer who participated in BASCH testing that prototyping should be used early in a development project because of the risk-mitigation benefit (Walker, 1992).

By applying prototyping, the developer can quickly produce a working model to provide a clearer understanding of a system's behavior. After the feasibility of replacing the paper acquisition chart with computer-driven displays was determined, it took less than six months to build the prototype ASTRO system, including roughly 20,000 lines of new and reused software code. Several demonstrations were performed for schedulers to begin the feedback process. The demonstrations quickly proved the feasibility of producing the schedule on a computer-driven large-screen display. Unisys noted that "the prototyping approach allowed this conclusion to be reached in just a few months, including several iterative generations of development, demonstration, and revision" (Unisys, 1988a:20). Additionally, "the ASTRO prototype also contains the fully functional database and library of utility routines necessary to support both the display study and an operational high-performance automated scheduling system" (Unisys, 1988a:5).

The ASTRO development team was able to take advantage of a "high degree of portability for the scheduling [application program] designs, from SCRABL II and CPCI 350" which contributed largely to the 20,000 lines of code (Unisys, 1988a:17). This was cited as a major contributing factor to the high code productivity. The developer used mostly Turbo Pascal,

supplemented with C, Fortran, and Assembler to expedite code generation and revision. Additionally, they used the MS-DOS operating system and commercially available peripheral devices, drivers, and utilities for the demonstrator. After many refinements and additions the original system has evolved into the current ASTRO system.

Prototyping allows flexibility in selection of hardware and support software for the operational system. The ASTRO developer was never hindered by specifications or the prospect (time and cost) of changing documentation when new hardware or software alternatives were considered. The ASTRO prototypes were used to evaluate many different hardware devices, some of which were never used in the final system. For instance, a touch-screen device for the Greyhawk large-screen display was originally proposed but was dropped when an alternative screen selection device, the sonic pen, proved to be superior. The system configuration remained flexible right up to the installation of the fully operational ASTRO network in 1992.

To a large degree, the evaluation of hardware alternatives was the intent of the original study. The initial feasibility study investigated several technologies for achieving the computer-driven large-screen display objective, including raster-scan and vector-scan CRTs and laser projection screens. The very high resolution and very large screen requirements derived from the analysis of the paper chart were "at the fringes of the current 'state-of-the-art' of display technology" (Unisys, 1988:2). During

June and July 1991, Unisys conducted a lab evaluation of a new Sony CRT display and a Metheus Corporation display controller to replace the Greyhawk display, concluding that the system should be upgraded to the new display. "Conversion of the ASTRO software to operate with the Sony CRT and Metheus display controller required only three weeks" (Unisys, 1991:18).

The tenth advantage of prototyping is that it can produce a better user-system interface for large multi-user systems. This advantage is not clearly evident because ASTRO was developed for a single-purpose single-mission environment. ASTRO was also a relatively small software development effort.

Finally, prototyping provides higher probability of delivering the "right" system the first time. In this one case, the prototyping attempt did deliver the right system while the traditional development attempts failed. But probabilities cannot be determined from a sample of one. Also, the definition of "right" system is subject to interpretation. The right system for scheduling might have changed the way scheduling was done entirely. But the objective of the initial ASTRO prototyping effort was to re-host the activities associated with the paper acquisition chart to a computer-driven large-screen display, leaving its contents and format intact. A better term might be the "objective" system.

The analysis of the research data also identified two additional advantages of the prototyping methodology not discussed in the literature

review. In the ASTRO project, prototyping expedited the transition and integration of new technologies and commercial products into a computer system. Although this advantage is somewhat related to the flexibility in selection of hardware and support software, it warrants explicit mention because sometimes, as in the ASTRO case, it is the objective of the development effort. The ASTRO project quickly and successfully exploited new technologies in large screen displays, voice input/output equipment, screen selection devices, and commercial software applications.

The ASTRO project also benefitted from high programmer productivity throughout the development effort. For example, during a four month period, the developer required only 12 man-months of effort to write or modify in excess of 20,000 operational source lines of code (SLOC). Over the entire second phase, January 1988 to February 1989, the staff developed or modified nearly 45,000 SLOC in just 34 man-months or approximately 1325 SLOC per man-month (Unisys, 1988a:5). The ASTRO developer's productivity far exceeded industry standards for DOD projects and provided further justification for the continuation of the evolutionary prototyping methodology.

In his book, Capers Jones notes that an average programmer develops approximately three function points per man-month for DOD embedded software systems (Jones: 1991:147). Using Jones's nominal value of 71 Ada SLOC per function point for DOD software, a programmer on average produces 213 SLOC per man-month (Galorath, 1992:18-21). Jones

added that the low productivity rate for DOD projects was due in part to the complex nature of embedded software, as well as the constraints associated with DOD-STD-2167A and the enormous overhead required to develop the documentation and conduct the formal reviews (Jones: 1991:148). Thus, the ASTRO project exhibited over a sixfold increase in programming productivity as compared to industry standards for traditional DOD software efforts.

The literature search also identified ten disadvantages or potential pitfalls of prototyping which were listed in Table 2, page 27. An analysis of the ASTRO effort with respect to these ten potential problems follows. Again, specific examples and supporting information from the data are used.

The first listed disadvantage of prototyping is that complete requirements definition often occurs late in the development process. The ASTRO prototype development is no exception if requirements definition means in the formal, written specification sense. However, one of the primary purposes of prototyping is to define users' requirements in a physical, tangible sense, so it is arguable whether this is truly a disadvantage or merely a difference from traditional development. As cited earlier, the ASTRO developer acknowledged the risk to both the Air Force and developer of not having the accountability that specifications afford. The developer recommended that, in the absence of formal development specifications, "some form of 'Memorandum of Agreement', or a level of

effort contract with measurable milestones" be written to protect both parties. This recommendation was never implemented (Unisys, 1988a:7).

The study tasking required, and Unisys delivered, a draft specification of "the database, display formatting, and interactive schedule manipulation capabilities necessary to meet Resource Scheduling operational requirements" (Unisys, 1987:27). Unisys later recommended that generation of detailed specifications and system documentation be deferred until the prototyping effort was completed and then document it "as installed" (Unisys, 1988a:7). In July 1990, The Aerospace Corporation recommended that ASTRO be made operational and brought into conformance with DOD-STD-2167A regarding requirements specifications, documentation, testing, and security (Miller and Springsteen, 1990:3-1).

Another disadvantage is that the prototyping process lends itself to growth in system requirements through iterative users feedback. This is the problem of not knowing when to stop the iterations and conclude the prototype effort by formally documenting the requirements of the final system. In the ASTRO prototyping effort, it is difficult to distinguish outcome of the requirements definition process from requirements growth without a requirements specification baseline. The dividing line between when the system was a prototype and when it became operational is fuzzy. Also, the objectives of the ASTRO development were expanded midway through the effort.

When all of the study development goals had been achieved and the prototype deemed complete in February 1989, the ASTRO software consisted of 44,838 lines of code. Before operational evaluation began, Unisys commented that "at system demonstrations and technical interchange meetings, the schedulers and ASTRO development staff have discovered new Network Display enhancement ideas and automation opportunities nearly as quickly as the original goals have been met" (Unisys, 1989a:5). At that time, 17 additional software enhancement possibilities were identified for future work. During the following period ending 30 October 1989, a total of 48 software enhancements were actually implemented and the software had grown by 19,147 lines of code or 43 percent (excluding the network software, which was added scope) (Unisys, 1989a:28-30; Unisys, 1989b:15-31). Additionally, there were 10 hardware enhancements performed during the period. Most of the enhancements after this period were related to the expanded scope developments, and an ASTRO Software Specification was submitted in May 1991.

A third disadvantage of prototyping is that it may be slow to identify an optimum design. Throughout the ASTRO development, the system design evolved right up until the system's completion in July 1992. The basic architecture of ASTRO never changed, but individual hardware and software components have been improved, or optimized, with advances in technology. Therefore, the system did achieve an optimum design early and "upgraded" the component capabilities.

The next disadvantage is that a prototype effort may result in an inefficient system design. This problem results from the high-level, less-efficient programming languages typically used for fast prototyping. Recognizing this fact, the ASTRO developer decided at the outset not to use a single high-level language. Instead, they used a combination of Turbo Pascal, C, Fortran, and Assembler languages to develop custom database and display management software.

The developer also chose a dedicated, single-tasking operating environment (Microsoft MS-DOS running on a PC) to avoid performance problems and overhead of multi-tasking systems (Unisys, 1987:24). The developer noted performance problems in the BASCH and CCS system where "reliance on general purpose database management software, display management software, and the services of a commercial multi-tasking operating system which was not designed to support real-time operations" (Unisys, 1988a:15). Performance measurements yielded "system response times which are equal to or better than those of the operational SCRABL II system and which are far superior to those provided by the CCS mainframe computers" (Unisys, 1988a:17).

An oft cited concern with prototyping is the potential risk that the prototype system will be forced into serving as the operational system. In December 1987, Unisys stated:

> The final goal of this study is not to develop an operational scheduling system, but to explore alternatives, try out ideas, and evolve a set of specifications to be used for procurement of

> the next generation of scheduling hardware and software. (Unisys, 1987:27)

Despite this declaration, this pitfall was not avoided in the ASTRO development. Because of ASTRO's progress coupled with BASCH's stagnation, the users requested installation and operational activation of the "interim" ASTRO system after it had evolved to meet or exceed nearly all of the functional evaluation criteria for the scheduling systems.

The sixth disadvantage of prototyping listed is that it may produce a system which lacks portability or generality. Portability of the ASTRO system was limited to PCs running the MS-DOS operating system. "To maintain a degree of hardware independence, all of the ASTRO software is designed to run under the commercial MS-DOS operating system" (Unisys, 1988a:15). The system is not generalizable to other applications because the display management, database management, file-server, and network software was custom designed and tailored to the AFSCN range resource scheduling problem domain.

The seventh disadvantage notes that prototyping is not appropriate for large scale developments which apply parallelism or distribution of effort. Except for the file-server, the ASTRO system does not apply parallel or distributed architectures. At less than 100,000 lines of code ASTRO is not considered a large scale software development effort.

The eighth disadvantage is that it can be difficult to maintain a tight development schedule. The ASTRO study did not have development

milestones or schedule requirements. However, the prototyping effort did adhere very closely to the schedules and objectives which were set out in each of the progress reports. The completion of the prototype system occurred on 13 February 1989, just 44 days beyond its target date.

The next disadvantage relates to the fact that prototyping efforts generally do not provide clear, formalized deliverables for management to track development progress. This is certainly true in the ASTRO development, but it was by design. In the initial study tasking, there were only three deliverables required. The first was a demonstration of the prototype to the Air Force. The second was delivery of all of the source code to the AFSCN software library. The third was generation of a progress report for the task order period. The ASTRO project manager maintained control by keeping himself and the Air Force program manager closely involved in the prototype development. There was no traditional way to measure progress because of the informal nature of the development effort.

Lastly, in prototype efforts, documentation tends to lag behind development. This was also the case in the ASTRO effort. Again, this was done intentionally, as discussed earlier, to allow the prototyping effort to evolve and not get bogged down in specification and other documentation changes. A draft users' guide was delivered in July 1989 and was updated periodically thereafter. In July 1990, The Aerospace Corporation cited this documentation deficiency (for a system which would become operational)

and recommended that the ASTRO software and hardware be documented in accordance with DOD-STD-2167A and MIL-STD-490, respectively.

Investigative Question 7

What other factors might have influenced the contrasting outcomes of the two efforts?

This question addresses other factors identified during the analysis which may have influenced the contrasting outcomes of the two projects. Since the written documentation and archival records did not address these factors, the determination as to whether a factor impacted the outcomes was based primarily on the discussions with key individuals, as well as the personal insight of the researchers. Table 7 identifies the factors noted during the analysis.

The first factor identified during the research is the contrasting number of development steps associated with each effort. An Aerospace Corporation systems engineer suggested the BASCH effort attempted to resolve two major problem areas (steps) with a single development approach. He noted the initial BASCH design focused on two problem areas: make scheduling a less labor-intensive process and introduce automation into the process (Simpson, 1992).

The system concept for BASCH called for operational scheduling to rely on automation (i.e., computer algorithms) to perform the majority of the scheduling activities. The concept envisioned BASCH automatically

Table 7.

Contrasting Factors in the Two Projects

1.	One step development approach in BASCH versus two step approach in ASTRO.
2.	BASCH system prematurely rushed from development facility to operational environment.
3.	Contract type--formal contractual action versus flexible task order.
4.	Air Force management highly involved in the ASTRO project yet provided development flexibility.
5.	Sense of competition among the ASTRO development staff.
6.	Stagnation in BASCH development supported the continued development of ASTRO.

handling most day-to-day scheduling activities, freeing a smaller scheduling staff to deal with real-time conflict resolution. However, funding constraints resulted in deletion of the automation requirements from the BASCH system specifications. As a result, the delivered BASCH provided a scheduling system with two databases, resident on geographically separated computers, connected via an internode communications link. Furthermore, scheduling remained a highly labor-intensive process requiring the paper acquisition chart to be maintained at both RCCs in order to support the dual-node operations concept (Simpson, 1992).

In comparison, the Aerospace Corporation systems engineer suggested that the ASTRO development was accomplished in a two-step approach.

First, the developer produced the software to allow the schedulers to use a single computerized database to conduct scheduling operations rather than the paper chart. Once this was accomplished to the satisfaction of the users, the developer generated additional system capabilities to provide computer processing of certain scheduling activities. These capabilities reduced the number of activities requiring manual processing effectively reduced manning requirements (Simpson, 1992).

The second factor relates to the perception of the Air Force BASCH test team that the system was prematurely rushed from a development environment to the operational environment. The developer's facility lacked an internode test-bed, which meant the BASCH system was delivered without formal testing of the internode data transfer function. As discussed previously, problems with BASCH started with system test of the internode function (Walker, 1992). In addition, the Aerospace systems engineer suggested that the transition of BASCH to an operational environment reduced the developer's ability to correct software deficiencies in a timely manner. Since the system was operational, deficiency corrections could only be delivered via the formal maintenance process. In contrast, if the system had remained in the development environment, correction could have been more quickly and effectively incorporated into the system (Simpson, 1992).

In comparison, the ASTRO system remained in a development environment right up to the delivery of the final software master. The final

master, Master #1.43, was installed in the ASTRO system on 31 July 1992 to support system turnover to AFSPACECOM (List, 1992).

The next contrasting factor relates to the methods used to develop the two systems. The Air Force procured the BASCH system via a formalized cost-plus contract option. This approach provided little flexibility in the event the government needed to make changes to the baseline system. Additionally, the program office lacked a means to provide feedback directly to the developer, since the effort was conducted outside the scope of the contract award fee plan. In comparison, the ASTRO effort was conducted under a yearly level-of-effort contract task order. This approach allowed the SPO to redirect or expand the effort as required. It also provided the SPO with the option to terminate the effort if the developer failed to meet desired project objectives. Either of these actions could be easily accomplished through a routine contracting officer letter. The approach also provided a direct feedback link with the developer through the contract award fee evaluations.

The fourth factor pertains to the impact the Air Force program management style had on the ASTRO effort. It was noted that the ASTRO program manager took a proactive role in the project while maintaining a positive government/developer relationship. The program manager also strongly supported the users' requirements and ensured the ASTRO effort continually progressed toward meeting those requirements. While maintaining an active interest in the project, the manager also provided the

necessary flexibility required to assure project success. Furthermore, a single Air Force program manager directed the ASTRO development from project start, October 1987, until the interim operational installation, January 1991. This provided continuity throughout the effort (Wong, 1992; List, 1992).

In contrast, The BASCH effort suffered from a lack of overall management attention, especially after the project was grouped with other CCS functions for software maintenance. In addition, over the duration of the development effort the BASCH project had numerous Air Force program managers. Each manager, within the constraints of the contract, tried to direct the effort toward their perceived system objectives. Thus, the BASCH developer suffered from inconsistant guidance, which further convoluted the effort (Wong, 1992).

The fifth factor relates to the competition perceived by the ASTRO development staff between ASTRO and BASCH. The ASTRO developer realized early in the range scheduling automation study effort that success of BASCH would most probably lead to the cancellation of the study effort. The ASTRO staff accepted the challenge of this situation and used the threat as motivation to drive the project toward success. The ASTRO project manager also comprehended the nuances of the acquisition environment. He observed that if ASTRO was to be accepted by the operational community, it would not only have to match the capabilities of BASCH but must far exceed them (List, 1992). This was due in part to the

perception among higher Air Force management that the sunk cost of BASCH must be justified, even though it meant pouring more money into the failing BASCH effort (Wong, 1992).

The final factor involves the continuous stagnation of the BASCH project, while the ASTRO effort continually improved and evolved. As mentioned previously, the ASTRO development staff felt their project would be terminated upon the successful delivery of BASCH. However, with each successive slip in the availability of BASCH, the ASTRO developer refined the system until the capabilities of ASTRO far exceeded BASCH and met the operational requirements of the schedulers. Furthermore, with each new ASTRO prototype model, the number of ASTRO supporters grew, eventually becoming a force within HQ AFSPACECOM which could no longer be ignored (List, 1992).

In summary, the analysis identified several additional factors which may have impacted the contrasting results of the two efforts. These factors may have directly or indirectly influenced the project outcomes. However, as with the three factors identified in Investigative Question 5 which were not advantages of prototyping, these contrasting factors only enhanced the benefits gained by using evolutionary prototyping in the development of the ASTRO system. In comparison, the contrasting factors identified above only provided more problems which the BASCH development effort would have had to overcome in order to reverse the inevitable outcome.

Summary

The analysis so far has focused on a single purpose, that being answering the investigative question individually based on the three data sources. So far no attempt has been made to draw any overall conclusions based on the findings of each question. Chapter VI provides a summation of the analysis and ties the findings of these questions together to answer the overall research question. Additionally, Chapter VI proposes areas of future research relevant to this case study.

VI. Conclusion

The conclusions of this research are: 1) although started at different times and with distinct objectives, the ASTRO and BASCH projects addressed the same basic problem; 2) the principal reason for the ASTRO project's success is application of the prototyping development methodology; and 3) the reasons for the BASCH project's failure are not limited to problems associated with the traditional waterfall development methodology. Additional factors which contributed to each project's outcome were identified. In the ASTRO project, these factors were complementary to the prototyping methodology, while some factors identified in the BASCH project were not at all related to the development method, and would have caused significant problems even if an alternative method had been employed.

ASTRO Success

Seven factors contributing to the success of ASTRO were identified in the documentation, archival records, and interviews with key participants. Selection of the prototyping method at the outset of the range resource scheduling automation study was clearly the most significant factor in the eventual success realized by the ASTRO project. As discussed in Chapter V, four of the seven factors can be directly attributed to theoretical advantages of prototyping identified in the literature review. The other three factors

were development team strengths which facilitated the effective use of prototyping. Those factors were the domain experience and knowledge of the developer's team, lessons learned from the previous scheduling systems (e.g., BASCH and SCRABL II) and the initiative and commitment of the development team to the process and product.

Ten of the eleven theoretical advantages of prototyping listed in Table 1, page 24, were supported by this case. This includes the four advantages explicitly identified in the data as factors contributing to the success of ASTRO. The remaining six advantages were born out through the data analysis. The one advantage not supported by the data is the applicability of prototyping to the development of large multi-tasking systems. However, since ASTRO was never intended to operate as a large multi-tasking system, it could not be expected for the data to reflect this advantage. The data analysis also identified two additional advantages of prototyping evident in the ASTRO project, which were not discussed in the literature review: quick technology exploitation and higher programmer productivity. Technology advances are expedited with prototyping because the requirements and design are not bound by specifications which may be too rigid or may contain preconceived solutions based on existing, proven technology. Furthermore, higher productivity is attained when the developer is not burdened by the large documentation and formal review overhead associated with the traditional development approach.

The principal advantage realized through prototyping was in the area of requirements analysis. The inherent synergy of prototyping facilitated identification, definition, clarification, and incorporation of requirements which had eluded previous attempts at automating the AFSCN range resource scheduling activity. The requirements were gradually extracted from the users with successive iterations of prototype fabrication and users' evaluation. The resulting system exhibited characteristics and functional capabilities which the schedulers took for granted in the manual system, but had previously been unable to communicate to developers in formal requirements specifications.

Of the ten theoretical disadvantages of prototyping identified in Table 2, page 27, only one is considered consequential in the ASTRO development. In fact, some of the theoretical disadvantages are simply differences from the waterfall development method and are not necessarily problems. An example is the late definition of requirements in prototyping efforts. This is by design and is not considered a disadvantage; rather, it is considered an advantage which allows the requirements to evolve with the prototype.

The one disadvantage evident in the ASTRO project relates to the decision to make the proof-of-concept ASTRO prototype the near-term operational scheduling system, replacing BASCH and SCRABL II. The theoretical disadvantage notes that forcing the prototype system into serving as the operational system results in a potential increase in risk. In

this specific case, the risk relates to potentially significant costs of maintaining the prototype system in the operational environment. The initial ASTRO prototype was not developed with the prospect of becoming an operational system, so supportability issues and associated costs were not considered in the trade-offs of programming languages, documentation, and commercial hardware and software. However, because the system became operational, the trade-off became one of making the system conform to standards or retaining the developer for sole-source support. Neither choice is without additional costs to the ASTRO project.

BASCH Failure

Use of the traditional waterfall method of computer system development in the BASCH project was only one of many significant factors which contributed to the system's failure to be accepted by its intended users. A total of eight factors which influenced the failure of BASCH were identified from the analysis of the three data sources. Three of the eight can be considered disadvantages of the traditional waterfall development methodology as opposed to prototyping. The other five factors were not directly attributable to the development methodology chosen. These factors were the result of Air Force and developer management decisions and the interdependence of BASCH with the rest of the CCS development.

These other factors, such as the geographical separation of the users from the analysts and programmers and the constraint to operate within

the CCS environment, would have severely crippled even a prototyping effort. It cannot be determined which factor, if any, had the most detrimental impact. But certainly the combination of these eight factors produced a development process and product that was doomed to failure.

Each of the six other contrasting factors, which were observed but not sufficiently supported in the data, warrant some consideration for future computer system developments. The first contrasting factor demonstrates the problems which can occur when the project is too ambitious and tries to accomplish everything at once instead of one piece at a time. The second factor illustrates the importance of testing in a high-fidelity test facility, if at all possible, before delivering software to users. The third and fourth factors indicate that management visibility and control can be achieved in less formal, more flexible contract types. Finally, the last two factors indicate that competition may actually further enhance the responsiveness and productivity of prototyping efforts.

Suggestions for Future Research

Unfortunately, a limitation of the methodology used in this study is that the conclusions reached are not generalizable to other DOD computer system development projects. Therefore, a research effort should be undertaken to use statistical methods to answer the general research question of what benefits are realized through prototyping and what are the types of projects which benefit most from the methodology. Another

approach would be to combine this case with other similar cases, perhaps in a meta-analysis design, to reach a more general conclusion.

The original intent of this research effort was to conduct a statistical study, using existing sources of computer system development project data, to determine if application of the prototyping methodology during the requirements definition and preliminary design phases of the software development life cycle leads to more successful outcomes. However, attempts to collect statistical samples of data were unsuccessful because the kinds of data needed are not routinely recorded and the measures are not standardized. In fact, some developers are reluctant to release such information even when confidentiality is maintained. To collect the data directly would have required more time than the researchers had available.

The population of interest is DOD computer system projects built since DOD-STD-2167A was published, when developers were given the option of using alternative development methodologies. A statistical comparison of two or more development methodologies could be performed using samples of measurement data from actual projects. One could determine the degree of success that the DOD has had with each methodology and develop criteria or guidance for selecting a development method for a particular project.

Appendix A: A Description of AFSCN Scheduling Operations

(Authored by John List, Paramax Corporation, Sunnyvale CA, Undated)

1. INTRODUCTION

This paper provides a brief generic description of current AFSCN scheduling operations. The steps necessary to build, publish, and maintain the schedule of network activities are described. The intent is to identify the major activities accomplished by the schedulers and the services provided by the scheduling computer system, allowing derivation of high level functional scheduling software requirements.

Scheduling consists of three major activities:

a. Schedule Construction - the "7-Day."
b. Conflict Resolution and Schedule Publication - the "Support Message."
c. Schedule Maintenance - "Real Time."

It is difficult for visitors to observe these three steps because all three are "in work" simultaneously. Thus, while some schedulers are maintaining the real time schedule, others are preparing tomorrow's support message, and still others are building next week's 7-Day schedule. For simplicity, the steps will be described sequentially.

2. 7-DAY SCHEDULE CONSTRUCTION

Schedule construction consists of gathering acquisition data for all spacecraft, gathering support and non-flight requirements from the Mission

Control Complexes (MCCs), tracking stations, and developers, building the Acquisition Chart and selecting support times, and publishing the 7-Day schedule.

2.1 DATA INPUT

The MCCs provide two basic inputs to scheduling: acquisition data and request data.

2.1.1 Acquisition Data

MCCs which have vehicles in non-synchronous orbits must supply the schedulers with the predicted visibilities (acquisitions)) for their satellites, since supports for the satellites can only be accomplished when they are visible from one of the tracking station antennas.

Normally, each MCC accomplishes its own ephemeris generation and provides the resulting acquisition data to scheduling by delivering a Satellite Acquisition Tape (SAT). SATs are loaded directly into the scheduling computer.

A "state vector" may be provided instead of pre-computed acquisitions on a SAT. In this case, ephemeris generation is accomplished off-line (by the Schedule Plans staff or by Inter-Range Operations) and the resulting acquisition data is transferred to the scheduling computer via SAT.

Acquisition data on a SAT may be modified when the tape is loaded into the scheduling computer system. For example, an "offset time" may be

applied to the data to reflect a change in launch time. Revolution numbers may be also be altered (removing the tenths digit, for example).

2.1.2 Request Data

All MCCs supply the schedulers with weekly Program Action Plans (PAPs) which describe specific support requirements for each satellite. While the acquisition data describes all the times and places each satellite could be supported, the PAP describes which of those support opportunities are needed for actual commanding, telemetry reception, and tracking. PAPs may be supplied to scheduling in several formats. Some MCCs provide their PAPs on magnetic tape in a machine readable PAP tape format. Handwritten paper PAPs are still used by some MCCs, while others use various computer systems and programs (including the Vehicle Acquisition List function of the scheduling computer itself) to produce PAP listings.

For synchronous vehicles, the PAP also supplies the schedulers with current visibility information. The schedulers use this information to update the scheduling computer database (either through generation of dummy acquisitions for the synchronous vehicles or through update of "synchronous flags" in the system environment tables.

Non-flight requests are submitted to the Schedule Plans staff. Remote locations submit their requests via "9-line" teletype message, while local non-flight requests may be submitted via paper forms or PAP tapes.

2.2 ACQUISITION CHART CONSTRUCTION

When the acquisition data has been loaded into the scheduling computer, the schedulers begin construction of the Acquisition chart. This activity starts approximately a week and a half before the beginning of the time-span being processed.

2.2.1 Plotting of the Acquisition Chart

An off-line Personal Computer and high-speed pen plotter are used to draw selected subsets of the acquisition data on the "Acquisition Chart".

Satellites in very low orbits, or those which may later change orbit are not plotted. Satellites which are infrequently supported are also omitted from the plot, as are synchronous satellites. The Global Positioning System (GPS) satellite visibilities are omitted from the plot because the large number of satellites in this family would overload the chart. Thus, plotting is essentially limited to medium and non-synchronous high orbit vehicles with the exception of GPS. Each Monday morning, the data for the next week is plotted, resulting in a chart that is 36 inches wide and 84 feet long.

2.2.2 Selection of Low-Orbit Supports

By the time plotting of the chart has been completed, PAP inputs have arrived and construction of the schedule begins. This activity starts with update of the chart and the computer database to reflect known low-

orbit support requirements. The procedure used to make these updates depends on the scheduling computer system in use, but the result is the same. At the Acquisition Chart, an "M" is drawn behind each requested support. The data system to be used for support is indicated by plotting a color-coded tape beneath the support on the chart. The computer database is then updated to show that these acquisitions will be supported.

2.2.3 Selection of Major Downtimes

Next, the Schedule Plans staff manually plots major downtime blocks. These blocks represent times when a tracking station resource is unavailable due to a major maintenance or modification effort, and when contracts or other limitations make it extremely unlikely that satellite operations will be able to bump the downtime effort. Such major downtimes are pre-coordinated with the MCCs at a weekly long range scheduling meeting (held the previous Friday), in an attempt to keep the MCCs from requesting resources which are already known to be unavailable. These events are plotted on the Acquisition Chart in wide yellow tape, with annotations describing the downtime written directly onto the tape.

2.2.4 Selection of High-Orbit Supports

Supports for high orbit vehicles (both synchronous and non-synchronous) are now selected (Monday afternoon and all day Tuesday). Again, the procedures used depend to some extent on the scheduling

computer system in use and also vary depending on whether a paper PAP or a PAP tape have been provided to scheduling.

Supports are manually selected at the Acquisition Chart. Each selected support is plotted using colored patterned adhesive tapes (the pattern indicates the spacecraft family, while the color indicates the specific satellite). The schedulers work at the chart using a listing of the support requests provided by the PAP.

Each requested support is selected and plotted on the chart using a conflict avoidance philosophy. Within the time and visibility constraints outlined on the PAP (requested times) and Acquisition Chart (visibility information), the scheduler attempts to find a reasonable way to fulfill the PAP request. The scheduler also attempts to avoid previously selected major downtimes and flight supports, while also considering future inputs not yet received. Since many low orbit MCCs have not submitted detailed PAPs at this point, and none of the MCCs are willing or able to formally resolve conflicts this far in the future, the schedulers do not waste effort on optimization. Instead, the desire is to get as much raw data onto the Acquisition Chart as possible to facilitate later conflict resolution. Conflicts, when discovered, are simply ignored.

To prepare for later conflict resolution and schedule maintenance activities, a condensed summary of the PAP constraints (which the schedulers call a "PAP Window") is written on the chart below each selected support. The PAP Window annotations allow most later schedule

manipulations to be accomplished without the need to consult the PAP. As with low orbit supports, the data system to be used is indicated with a strip of color-coded tape below the support itself.

2.2.5 Selection of Routine Downtimes

After the high orbit supports have been selected, the Schedule Plans staff returns to the Acquisition Chart to manually select schedule times for routine downtimes and maintenance. This is normally accomplished late on Tuesday or early Wednesday morning and is an interactive process between the schedulers and the planners. In theory, the routine downtimes will be selected in "holes" left after flight supports have been picked. In actual practice, there are always insufficient "holes", so the high orbit supports require adjustment to accommodate downtimes. As with major downtimes, these non-flight activities are plotted in wide yellow tape and are annotated to provide a description of the activity.

2.3 UPDATE OF THE COMPUTER DATABASE

Wednesday 7-Day efforts are devoted to update the scheduling computer database to reflect the high-orbit and downtime supports previously selected at the chart. The specific procedures used to update the database are dependent on the computer system in use and may include coding of keypunch forms and/or either on-line or off-line interactive task creation. When all of the selected supports have been entered in the

database, Site Schedule Lists are obtained to cross-check the database against the contents of the Acquisition Chart. Interactive database updates are then accomplished to correct any errors found.

2.4 PUBLICATION OF 7-DAY SCHEDULE PRODUCTS

On Thursday morning, the scheduling graveshift publishes the weekly Frequency Protection Messages to supply satellite visibility and radio frequency utilization information to organizations responsible for frequency management near the VTS, HTS, and GTS remote tracking stations (RTSs). The AUTODIN output capabilities of the Vehicle Acquisition List function are used for this publication. The frequencies used and turnaround times required for each vehicle are included on this message through the use of "Text Headers" contained in the scheduling computer database.

Thursday at noon, the Schedule Plans staff publishes the 7-Day schedule containing (only) non-flight and rehearsal activities. The 7-Day is routed to each RTS, MCC, and to contractors who have requested downtimes. The AUTODIN output capabilities of the Site Schedule List function are used here.

The scheduling swingshift (Thursday evening) uses the Site Schedule List function to publish the 7-Day In-House listings of flight supports scheduled for selected MCCs. These are printed listings and magnetic tapes for hand delivery, not AUTODIN outputs.

All of the published 7-Day schedule products take effect the following Monday at 0000Z.

When all 7-Day schedule construction efforts are complete, the Acquisition Chart is physically moved from the schedule planning area to the real time scheduling area and construction of the next 7-Day schedule begins.

3. CONFLICT RESOLUTION AND SUPPORT MESSAGE PUBLICATION

Each day the schedulers must produce a 24-hour schedule for all the Air Force Satellite Control Network (AFSCN) operating elements. At the end of the day, a conflict free schedule must be completed for transmission to all network elements. The daily activities to produce the schedule are divided into three major parts: To-Do, Conflict Resolution, and Schedule Publication.

3.1 TO-DO ACTIVITIES

Daily conflict resolution efforts begin late on dayshift two days before the schedule is to take effect. This activity is known as the "To-Do" (named after the clipboard where changes for future time-periods are kept until sufficient Acquisition Chart is unrolled to reach the supports).

During the To-Do effort, the Acquisition Chart and scheduling computer database are updated to reflect "final" complete support

requirements for all vehicles. This includes update of satellite visibility times and support requirements based on latest ephemeris runs and MCC support plans including schedule change requests provided via revised PAPs or Manning Schedule Change (MSC) forms.

To-Do chart work begins with the plotting of visibilities for very low orbit satellites. These vehicles are not plotted on the chart for the 7-Day because they tend to do orbit adjusts which cause their visibility times to change. Next, an "M" is written behind each of these "low flyers" which is to be supported. The support requirements (including pass supports, pre-passes, command messages, and playbacks) are annotated on the Vehicle Acquisition List used for plotting. This annotated list is then used to update the scheduling computer database.

Next, the plotting accuracy on the chart is checked for vehicles which were plotted on the 7-Day. This is accomplished in three steps. First, a Vehicle Acquisition List is compared to the latest ephemeris run to see if there is a significant difference in visibility times. Then the acquisition data in the computer database is adjusted as required using the Slide Tasks function. Finally, the corrected data is listed using Vehicle Acquisition List and compared to the Acquisition Chart plotting. Any differences are corrected on the chart. This entire procedure is known as "arrowing", or "updating Acquisitions".

Since some of these vehicles do not supply an accurate PAP for the 7-Day, their support requirements are now plotted on the chart with an "M"

behind each requested visibility. The scheduling computer database is also updated to reflect these support requirements.

When "arrowing" is completed, any pending schedule change requests (MSCs or revised PAP requests) are processed. These change requests represent new or changed requirements submitted by the MCCs in the interval between construction of the 7-Day schedule and the start of the To-Do effort. Each schedule change is annotated on the Acquisition Chart and the database is updated to reflect the change. The chart annotations for these events include the date and time at which the change was formally received by scheduling, called the Date/Time Group (DTG). The DTG is written, enclosed in a circle, on the chart near the event(s) modified by the MSC.

When all updates to the database have been completed, a Site Schedule list is run to compare with the Acquisition Chart. This procedure is intended to cross check the work at the chart and in the computer database to make sure both representations of the schedule are the same.

All the above activity is concerned with the schedule two days in the future. When the To-Do activities are completed, the Acquisition Chart contains all known requirements for use of AFSCN resources for that 24-hour time span. No conflict identification or resolution has been performed.

3.2 CONFLICT RESOLUTION ACTIVITIES

3.2.1 The Shuffle

When the To-Do activities are completed, resolution of conflicts begins. The schedulers manually examine the Acquisition Chart for the To-Do period just completed and attempt to minimize the number of conflicts through adjustment of support sites or times within the constraints dictated by the PAPs. This process is known as the "Shuffle" or board cleanup." This is a strictly manual effort at the Acquisition Chart, but this process generates many changes to the schedule which must then be transcribed into the computer database.

The computer database is updated after the shuffle has been completed. These database updates consist of "ADDs" and "VICEs". ADDs are new entries into the system. The satellite acquisitions have been previously loaded as part of the 7-Day cycle. Now, support periods are ADDed to the existing acquisitions. VICEs are changes to supports which have already been input as part of the 7-Day or To-Do cycles. These changes are annotated on Site Schedule Lists generated from the computer database. All changes are input to the computer system using task creation functions (ADDs) and interactive task update functions (VICEs).

The schedulers coordinate the results of the shuffle with each MCC and explain remaining "hard" conflicts. These are the conflicts which can only be resolved through MCC action (deletion of support, changing of PAP

requested times or stations, acceptance of non-standard hardware configuration, etc.). The schedulers aid the resolution of these conflicts by suggesting reasonable support alternatives based on experience, but the MCCs have final responsibility for their resolution, which may be based on classified mission objectives of which the schedulers have not knowledge.

As the "hard" conflicts are resolved, the resulting schedule changes are reflected on the Acquisition Chart and through update of the scheduling computer database.

3.2.2 First Board Check and Conflict Scan

Once the post-shuffle schedule changes are entered into the database, a Site Schedule List is generated for comparison with the Acquisition Chart. At this point, the Acquisition Chart is not completed. The schedule is not yet completely conflict free, but the Acquisition Chart is always "ahead" of the database resulting in additional ADDs and VICEs.

When these updates are completed, the schedule depicted on the Acquisition Chart and in the database should be the same. In actual operations, however, this is rarely 100% true, since the schedule continues to change while the checks and database updates are being accomplished.

To verify the quality of the schedule, the Conflict Scan function is now run. This function identifies any remaining conflicts (time/equipment overlaps between tasks). Conflicts thus identified are the result of data entry errors, miss-plots on the Acquisition Chart, or they may be legitimate

non-yet-resolved conflicts. The conflict list is checked against the Acquisition Chart and corrections are made as required to both the chart and the database.

3.2.3 RFI Conflict Identification

The prediction of RFI conflicts can be divided into two parts. First, an off-line computer run is made which calculates conjunctions (close approaches, as viewed from the tracking stations) between satellites. This calculation must be performed fairly close to real-time when accurate ephemeris data is available. Normally the runs are made by the Schedule Plans or Inter-Range Operations staffs on Monday, Wednesday, and Friday with each run covering the time span until the next run (plus some contingency overlap). During some critical satellite flights, runs are performed daily.

The second part of RFI prediction, actual identification of RFI conflicts, is performed daily by the schedulers and can only be accomplished when the schedule at the Acquisition Chart is fairly completed. Thus, RFI conflict identification is usually performed at the chart while the scheduling computer database is being updated to reflect changes identified by the first board check.

Identification of RFI conflicts is (currently) a manual task. The listing of all conjunctions between satellites with common radio frequencies is compared to the schedule as shown on the Acquisition Chart. The

schedulers draw an asterisk (*) above each task in conjunction. These flags allow quick identification of new RFI conflicts which might be created by later schedule changes. Conjunctions in which two (or more) vehicles plan to use a common radio frequency (both scheduled, or one scheduled and another with a "beacon" which is always on) are marked on the chart as RFI conflicts.

Supports in RFI conflicts which can be moved to alternate sites or times are moved to avoid the RFI. Supports which cannot be moved are defined as formal RFI conflicts. Some RFI conflicts require active resolution by the schedulers. For example, a call to NASA may be required to attempt to have a conflicting Landsat support moved. Other RFIs require only that the schedulers notify the affected MCC in writing that the RFI is predicted. The assumption is that the MCC will arrange support plans to work around the RFI problem (don't attempt critical data readout during the predicted RFI, for example).

3.2.4 Second Board Check

As soon as all first board check activities are completed, a second check is performed. A Site Schedule List is again compared to the Acquisition Chart. Procedurally, each scheduler is required to check different sites than those checked the first time.

While the listing is being compared to the chart, the scheduling computer system is used to begin preparation of data system schedules,

including the Bird Buffer computer schedule and Range Controller schedulers for both the Sunnyvale and Colorado Range Control Complexes. These schedules cannot be prepared until the site schedules are relatively "clean".

The Second Board Check results in more updates to the database. These updates and the preliminary data system schedules are generally completed by about 1600L. Thus, by 1600L, tomorrow's schedule should be 100% conflict free and the scheduling computer database should match the Acquisition Chart.

3.3 SCHEDULE PUBLICATION ACTIVITIES

3.3.1 Final Schedule Checks

When the second board check activities have been completed, the schedule should be conflict free and nearly ready for publication. To verify that the Acquisition Chart and the computer database are both correct, several final checks are accomplished. These checks are normally accomplished by the swingshift scheduling staff (the previous checks are done by the dayshift crew), to ensure that the final check is accomplished by fresh people.

Several Site Schedule Lists are generated, for the final checks. One list is a basis time-ordered list of all scheduled events for each RTS. Additional lists are obtained using the Site Schedule equipment selection

options. These lists are used to verify correct data processing assignments, wideband data link assignments, etc. The listings are red-lined to reflect changes or errors, which are then transcribed to the computer database.

Once these "final" checks of the schedule are completed, the data system schedules are finalized.

3.3.2 Publication of the Support Message

At this point, the database is frozen while the 24-Hour Support Message is published. All the various schedules produced must be made from the same database to prevent schedule ambiguities. Schedule publication consists of approximately 60 listings from the scheduling computer. Separate Site Schedule Lists are made for each MCC and RTS. Additional listings are generated for use by the data system operators and the various communication system operators. Generation of the listings is streamlined by use of JCL or Macros to automatically supply the required list control parameters.

Listings for the RTSs and for some other external agencies are generated in AUTODIN format for transmission by the Air Force Communication Center. Listings for local organizations are produced both as paper lists and as magnetic tapes. Both are manually distributed.

The published Support Message takes effect the next morning at 0800L time. Thus, with the normal publication time of about 1900L, there is a 13 hour "pad" in the scheduling time line to allow catch-up of any

activity which has fallen behind. However, some of this pad is consumed by the time required for the Communication Center to transmit the schedule to the RTSs.

4. SCHEDULE MAINTENANCE

All shifts are responsible for maintenance of the schedule from the time it is published until the scheduled events happen. This is called real time scheduling. The schedulers are also the custodians of the schedule data after-the-fact. This "history" data is used to prepare statistical reports on network utilization.

4.1. REAL TIME CHANGES

Once the Support Message is published, schedulers must continue to monitor the schedule and respond to required changes. These changes can be due to schedule errors (incorrect requests from the MCCs or incorrect processing by the schedulers), vehicle problems, tracking station problems, or changing mission requirements. Typically, about a third of the scheduled activities are changed between publication of the schedule and execution of the activities in real time. Many of these changes are the result of a "ripple" effect: A single change in the middle of a tight schedule may require many additional changes to surrounding activities.

All real time changes are made first at the Acquisition Chart. Tapes are added, deleted, or moved to reflect the new correct schedule. If the

changes are for events within 8 to 12 hours of real time, all affected parties (RTSs, MCCs, ACES, Comm, etc.) must be notified by telephone. This is currently the most time-critical scheduling activity since the modified schedule cannot be implemented until those who actually perform the support are notified. The scheduling computer cannot assist in change notification because the schedule changes have not yet been input to the database.

When all necessary verbal change notifications have been completed, a DD-173 message form is prepared for AUTODIN transmission. This provides formal hard copy notification of the changes. When the message is transmitted, the Date/Time Group of the message is written on the chart next to each scheduled event modified by the message, providing an audit trail of all schedule changes made subsequent to initial publication of the 24-Hour Support Message. Photocopies of the messages are used by the schedulers to update the computer database.

4.2 HISTORICAL VALIDATION

When a published schedule (including all changes to it) has been completely executed, it is termed "history" by the schedulers. Each night, the graveshift compares the computer database to the Acquisition Chart for the "history" day just completed. This check verifies that all change updates were entered into the computer correctly, so the database accurately reflects what actually happened in the network.

Periods where resources were not available due to outage are entered into the database at this time. Then, the Conflict Scan function is run to verify that the history data is conflict free. The Support Summary function is then run to provide "quick look" flight support statistics by tracking station and by program. The summary is also used to provide network loading information for the daily commander's briefing. After the summary is completed, a copy of the database is archived to magnetic tape and the tape is removed from the system for safekeeping.

The Schedule Plans statistician transfers the history data to magnetic tape for transport to the off-line statistical analysis archives, and finally, the history is deleted from the scheduling database. This history "purge" is performed every day to keep the active database as small as possible.

5. ADDITIONAL FUNCTIONAL REQUIREMENTS

As described above, both the Acquisition Chart and the computer database play a critical role in scheduling operations. The chart presents the "big picture" of the schedule necessary to quickly explore options and alternatives when manipulating the schedule. Further, the chart is immune to component failures, power or air conditioning failures, and software bugs. Thus, the chart provides insurance for the necessary continuity of scheduling operations. The scheduling computer and its database, though less "bullet proof" than the chart, provide valuable accounting, error checking, and schedule dissemination services not provided by the chart.

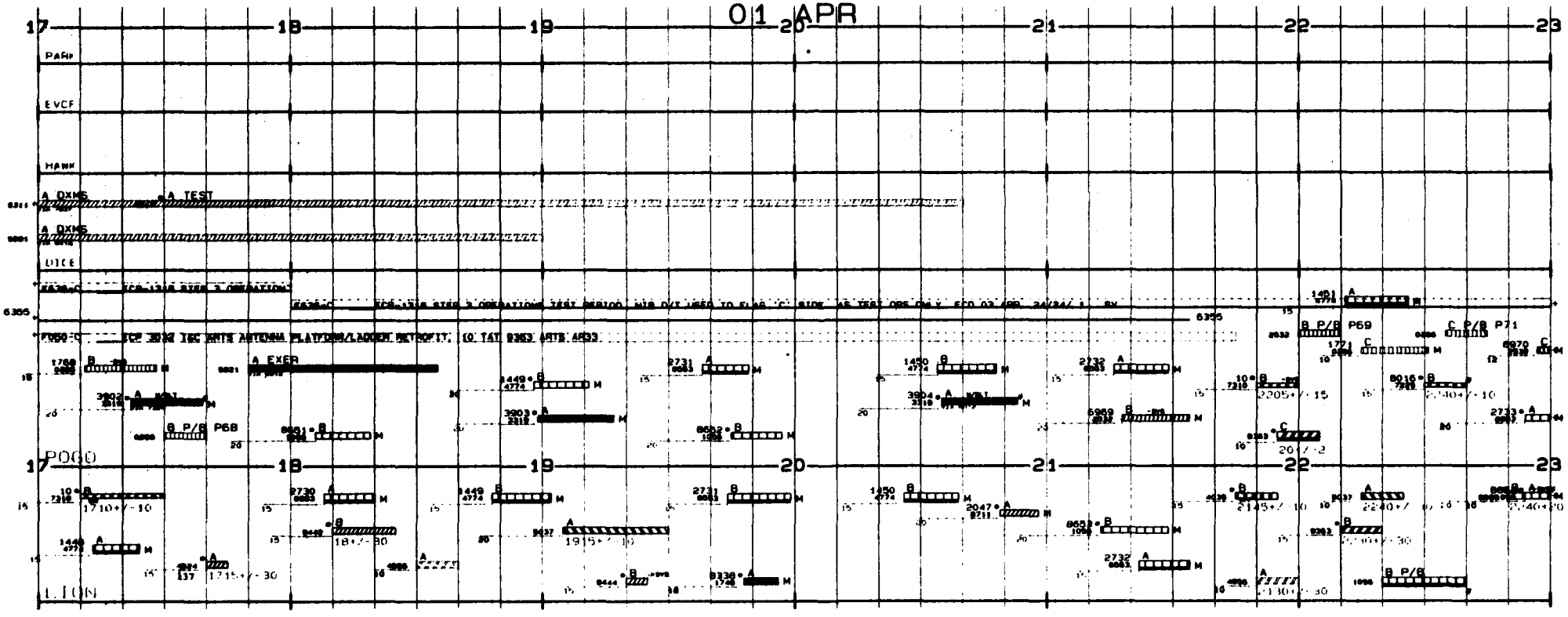
The combined capabilities of both of these scheduling "tools" are essential to successful scheduling.

At any time, the computer database represents a relatively volatile accumulation of the results of all scheduling work accomplished at the chart. Therefore, the schedulers are extremely careful to safeguard the data. Error checks are performed and non-volatile backup copies of the database are created several times per hour. An additional backup is physically removed from the system at the end of each shift to ensure that, in a worst-case computer, software, or operator induced "crash" only one shift of database updates would be lost. Safeguarding of the database is considered so important that turnover of the end-of-shift backup tape requires an initialed entry in the change-of-shift log book.

Appendix B: AFSCN Range Scheduling Acquisition Chart Facsimile

Attached is a facsimile of the range scheduling paper chart produced by an electrostatic color plotter and the ASTRO system database.

UNCLASSIFIED



Appendix C: Interview Preface

PROBLEM: As advances in technology increase the complexity of modern weapon systems, computers are called upon more frequently to control those systems. It can be said with a high degree of confidence that computers now play an integral role in virtually all major systems. However, the DOD faces a growing problem associated with the development of software: the inability of its managers to buy or develop quality software systems in a timely and cost-effective manner.

RESEARCH GOALS: Our goals in conducting this research are to explore the principal differences in the two system development efforts, BASCH and ASTRO, in the context of the unique AFSCN range resource scheduling problem domain, and to understand why the second project succeeded after the first was unsuccessful. The case presents a unique opportunity for comparison of the requirements definition and system development approaches used, because the nature of the problem did not change. Only the fundamental problem solving methods changed, within contractual and technological limitations at the time of the development effort.

RESEARCH DESIGN: The research design utilizes the case study method to explore the development processes of the BASCH and ASTRO efforts. We

will attempt to identify those factors which had significant influence on the results of the two projects. The interview responses will be combined with other data sources to address the following investigative questions:

1) What are the reasons cited for the non-success of BASCH?
2) What are the reasons cited for the success of ASTRO?
3) What development methodologies were used in each of the efforts? Why were the methodologies chosen?
4) How did the development methodologies used in the ASTRO effort enhance the process?
5) What were the perceived advantages and disadvantages of the development methodologies in this case?
6) What other factors might have influenced the contrasting outcomes of the two efforts?

INTERVIEW FORMAT: The purpose of the interview is to gather personal insight from key individuals associated with the BASCH and ASTRO development efforts. The interview will be conducted via telephone using an open-ended format so that additional areas of interest or insight can be pursued based on the individual's responses. The interview will be conducted in the following format:

I. Introduction.

II. Individual's Experience with the Problem Domain.

III. Individual's Role/Experience in the BASCH Development Effort.

IV. Individual's Role/Experience in the ASTRO Development Effort.

V. Investigative Questions.

VI. Additional Questions as a Result of the Individual's Responses.

VII. Conclusion.

Appendix D: Interviewee Background

Mr. John List, ASTRO Prototype Project Manager, Paramax Corporation - A Division of Unisys Corporation, Sunnyvale, CA.

Mr. List managed the ASTRO project from the initial proof-of-concept analysis, October 1987, through prototype system design and development, to installation of the fully operational ASTRO network in July 1992. From 1983 to 1987, he was lead programmer at System Development Corporation (now Paramax) assigned to SCRABL II system software maintenance. From 1968 to 1983, he was employed by the Air Force as a Master Scheduler at Sunnyvale AFS. In addition to his primary duties as an operational scheduler, he supported the development and evaluation of system-level and detail requirements for the BASCH system. From 1985 to 1987, he was also a member of the AFSCN Range Scheduling Working Group. Mr. List has a combined 24 years of AFSCN range scheduling experience as a developer and an operator.

Mr. Francis Wong, Scheduling Shift Supervisor, 21 SOPS/DOS, Onizuka AFB, Sunnyvale, CA.

Mr. Wong has 20 years operational AFSCN range scheduling experience. He joined range scheduling operations in 1972 after graduating from college with an Industrial Engineering degree. During his 20 years, he has progressed from Scheduling Trainee, to Master Scheduler, to Shift Manager, to his current position. In his current position, one of his primary functions has been as the direct interface between the range schedulers and the ASTRO development team. His duties included prioritizing ASTRO software deficiencies and suggested improvements, as well as reviewing associated documentation. He also was involved in the operational evaluation of both the BASCH and ASTRO systems and supported the BASCH effort as the users representative at the formal design/development reviews. In addition, Mr. Wong was a member of the AFSCN Range Scheduling Working Group from 1985 to 1987.

Mr. Bruce Simpson, Systems and Test Engineer, The Aerospace Corporation, Colorado Springs, CO.

From 1987 to 1991, Mr. Simpson was a technical advisor and system engineer for the Consolidated Space Operations Complex/System Program Office (CSOC/SPO) site activation task force (SATAF) organization. One of his primary responsibilities was monitoring the operational activation of the BASCH system. His duties involved observing and participating in BASCH testing, including Functional Qualification Testing, Component Testing, and System Testing. He also was the lead test engineer for the SATAF operational exercises of BASCH. In addition, he managed the resolution of BASCH software problems identified during testing. Finally, he was involved in the operational evaluation of the ASTRO prototype system at Falcon AFB in 1989. Mr. Simpson has over four years experience with the testing and activation of the BASCH system.

Mr. Rick Walker, Systems Engineer, International Business Machines Inc., Colorado Springs, CO.

From 1986 to 1989, Mr. Walker was the lead Air Force test engineer for the CSOC/SPO SATAF organization. His primary responsibility was supporting activities associated with the operational activation of BASCH. His duties involved observing and participating in BASCH testing, including Functional Qualification Testing, Component Testing, and System Testing. He also actively supported the SATAF operational exercises of BASCH. He was responsible for reviewing and validating proposed design changes to support deficiency resolution as submitted by the BASCH developer. He also reviewed associated BASCH enhancement documentation and preliminary ASTRO progress reports for HQ AFSPACECOM.

Upon separating from the Air Force in 1989, he joined IBM as a system engineer working CCS-related activities. In February 1991, he was assigned to the BASCH project as the lead system engineer. During this timeframe he supported the SSD/CW Near-Term Range Scheduling Requirements Assessment and was instrumental in the decision to activate the fully operational ASTRO system.

Bibliography

American Standards for Testing and Material. *Standard Guide for Rapid Prototyping of Computerized Systems*. ASTM E 1340-90. Philadelphia PA, 1990.

Benbasat, Izak, David K. Goldstein and Melissa Mead. "The Case Research Strategy in Studies of Information Systems," *MIS Quarterly*, 369-386 (September, 1987).

Boehm, Barry W. *Software Engineering Economics*. Englewood Cliffs NJ: Prentice Hall, 1981.

-----. "A Spiral Model of Software Development and Enhancement," *IEEE Computers, 14*: 61-72 (May, 1988).

-----, Terence E. Gray and Thomas Seewaldt. "Prototyping vs. Specifying: A Multiproject Experiment," *IEEE Transactions on Software Engineering, SE-10*: 290-302 (May, 1984).

Canan, James W. "The Software Crisis," *Air Force Magazine, 69*: 46-52 (May, 1986).

Carey, J, M. and J. D. Currey. "The Prototyping Conundrum," *Datamation, 35*: 29-33 (June 1, 1989).

Davis, Alan M., Edward H. Bersoff and Edward R. Comer. "A Strategy for Comparing Alternative Software Development Life Cycle Models," *IEEE Transactions on Software Engineering, 14*: 1453-1461 (October, 1988).

Department of the Air Force, *Acquisition Management: Life Cycle Management of Computer Resources in Systems*. AFR 800-14. Washington DC: HQ USAF, 29 September 1986.

Department of Defense. *Defense Acquisition*. DOD Directive 5000.1. Washington DC: Government Printing Office, February 1991.

Department of Defense. *Defense System Software Development*. DOD-Standard-2167A. Washington DC: Government Printing Office, February 1988.

Detachment 9, 2D Satellite Tracking Group (AFSPACECOM). *ASTRO Evaluation Final Report*. Falcon Air Force Base, Colorado Springs CO, 14 March 1990.

Egbert, D. E. and others. *Analysis of Man-Machine Interfaces for the Network Scheduling System*. Report to AFSCF/DVA. Sunnyvale Air Force Station, Sunnyvale CA, March 1986.

Eisenhardt, Kathleen M. "Building Theories from Case Study Research," *Academy of Management Review, 14*: 532-550 (1989).

Er, Meng C. "Prototyping, Participative and Phenomenological Approaches to Information Systems Development," *Journal of Systems Management, 38*: 12-16 (August, 1987).

Fruland, Lt Col William E., Commander .Letter to 2SWG/DOQ, "S/DR Fixes Required for Range Scheduling to Use BASCH as Operational System." Detachment-9, Falcon AFB, Colorado Springs CO, 18 September 1990.

Galorath, Daniel. Course handout distributed in SEER Users Course, Software Size Measures: Source Lines of Code and Function Points. ASD/FMC, Wright-Patterson AFB OH, May 1992.

Gavurin, Stuart L. "Where Does Prototyping Fit in IS Development?," *Journal of Systems Management, 42*: 13-17 (February, 1991).

Guimaraes, Tor. "Prototyping: Orchestrating for Success," *Datamation, 33*: 101-103+ (December 1, 1987).

Hager, James A. "Software Cost Reduction in Practice," *IEEE Transactions on Software Engineering, 15*: 1638-1644 (December, 1989).

Harker, Susan. "The Use of Prototyping and Simulation in the Development of Large-Scale Applications," *The Computer Journal, 31*: 420-425 (1988).

Holloway, C. Michael. "Analysis of Ada as a Prototyping Language," *6th AIAA Computers in Aerospace Conference, Technical Papers*. 1-10. Washington: American Institute of Aeronautics and Astronautics, 1987.

Jones, Capers. *Applied Software Measurement*. New York: McGraw Hill, Inc., 1991.

Kelly, Ronald and Richard Neetz. "Rapid Prototyping: The Procedure for Software," *Proceedings of the IEEE, NAECON 1988, Volume 2.* 644-651. New York: IEEE Press, 1988.

Klinger, Daniel E. "Rapid Prototyping Revisited," *Datamation, 32*: 131-132 (15 October 1986).

List, John. Untitled White Paper to SSD/CW, 4 December 1991.

List , John, ASTRO Prototype Project Manager. Telephone interview. Paramax Corporation, Sunnyvale CA, 27 July 1992.

Logemann, Col Dean D., Director. Letter to SSD/CW, "Range Scheduling Near Term Solution." HQ AFSPACECOM/ Force Support, Peterson AFB, Colorado Springs CO, 12 November 1991.

Lugi. "Software Evolution Through Rapid Prototyping," *IEEE Computers, 22*: 13-25 (May, 1989).

Marciniak, John J. and Donald J. Reifer. *Software Acquisition Management*. New York: John Wiley & Sons, Inc., 1990

Miller, Saul D. and Lois A. Springsteen. "AFSCN Range Scheduling System Assessment." *Report to* L. K. Konopasek. The Aerospace Corporation, Los Angeles, 23 July 1990.

Pagano, Col Vito J., Director. Letter to SSD/CWO, "Basic Scheduling (BASCH) and the Unisys/Mitre Resource Scheduling Research and Development Efforts." HQ AFSPACECOM/Space Systems Plans, Peterson AFB, Colorado Springs CO, 5 May 1989.

Pope, Capt Stuart. Internal Memorandum for Record, "Interim Scheduling Solution for Net 2/3." Consolidated Space Operations Complex System Program Office, Los Angeles AFB CA, 27 April 1990.

Purtilo, James, Aaron Larson and Jeff Clark. "A Methodology for Prototyping-in-the-Large," *13th International Conference on Software Engineering*. 13-25. Los Alamitos CA: IEEE Computer Society Press, 1989.

Rogers, Capt Richard L. *A Rapid Prototyping Approach to Software Validation*. MS thesis, AFIT/GCE/ENG/86D-10. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1986 (AD-A177939).

Rowen, Robert B. "Software Management Under Incomplete and Ambiguous Specifications," *IEEE Transactions on Engineering Management, 37*: 10-21 (February, 1990).

SATAF/TE and The Aerospace Corporation. "Summary Report for Phase II BASCH Exercise." Undated.

Simpson, Bruce, Systems Test Engineer. Telephone Interview. The Aerospace Corporation, Colorado Springs CO, 28 July 1992.

Smyrniotis, Chuck. "Rapid Prototyping: A Cure for the Software Crisis," *Proceedings of the 23rd Hawaii International Conference on System Science*. 202-210. Los Alamitos CA: IEEE Computer Society Press, 1990.

Staff Study. *Satellite Control Network Management: An Analysis of Programs Providing Common Multi-User Capabilities (Draft).* Washington DC: SAF/AQSS, 10 October 1991.

Swift, Michael K. "Prototyping in IS Design and Development," *Journal of Systems Management, 40*: 14-20 (July, 1989).

Systems Development Corporation. "Operational Suitability of DSM Basic Scheduling CPCI-350." Report to AFSCF/DVE. Sunnyvale Air Force Station, Sunnyvale CA, 13 March 1987.

Tate, G. "Prototyping: Helping to Build the Right Software," *Information and Software Technology, 32*: 237-244 (4 May 1990).

Unisys Corporation. "Task-56 Range Automation Study: Implementation Assessment for Range Scheduling via Large-Screen Displays, Phase One - Requirements Analysis and Equipment Recommendation." Report to SD/CLNND. Headquarters Space Division, Los Angeles, 7 December 1987.

-----. "Task-56 Range Automation Study: Implementation Assessment for Range Scheduling via Large-Screen Displays, Phase Two - System Prototype and Draft Specification." Report to SD/CWOD. Headquarters Space Division, Los Angeles, 30 June 1988a.

-----. "Task-56 Range Automation Study: Transmittal of Study Progress Report." Report to SD/CWOD. Headquarters Space Division, Los Angeles, 30 September 1988b.

-----. "Task-56 Range Automation Study: Transmittal of Study Progress Report". Report to SD/CWOD. Headquarters Space Division, Los Angeles, 13 February 1989a.

-----. "Task-56 Range Automation Study: Transmittal of Study Progress Report." Report to SSD/CWOD. Headquarters Space Systems Division, Los Angeles, 30 October 1989b.

U.S. Congress. *Bugs in the Program: Problems in Federal Government Computer Software Development and Regulation*. Subcommittee on Investigations and Oversight Staff Study. 101 Congress, 1st Session. Washington DC: Government Printing Office, 1989.

Voltmer, John G. "Selling Management on the Prototyping Approach," *Journal of Systems Management, 40*: 24-25 (July, 1989).

Walker, Rick, Systems Engineer. Telephone interview. International Business Machines Inc., Colorado Springs CO, 27 July 1992.

Weinberg, Randy S. "Prototyping and the Systems Development Life Cycle," *Journal of Information Systems Management, 8*: 47-53 (Spring 1991).

Whipple, L.K. and T.J. Hoida. "Rapid Prototyping as an Acquisition Strategy in the Air Force Satellite Control Network," *27th Annual Telemetry Conference*. Las Vegas NV, 1990.

Wong, Francis, Scheduling System Planner. Telephone interview. 21 SOPS/DOS, Onizuka AFB, Sunnyvale CA, 28 July 1992.

Wright, Capt H. G. Cameron and Capt Donald J. Aitken. "A Human Factors Approach to Range Scheduling for Satellite Control," *Proceedings of the Fourth Annual Workshop on Space Operations Applications and Research*. 484-489. Houston: NASA Press, 1990.

Yin, Robert K. *Case Study Research: Design and Methods*. Newbury Park CA: Sage Publications, Inc., 1989.

Vita

Donald J. Aitken was born on 12 November 1960 in Quincy, Massachusetts. He graduated from Weymouth North High School in Weymouth, Massachusetts in 1978 and attended Worcester Polytechnic Institute, Worcester, Massachusetts, graduating with a Bachelor of Science in Engineering Management in May 1982. In September 1984, he enlisted in the United States Air Force and received his commission through Officers Training School. Upon commissioning in December 1984, he attended the Air Force Communications Electronics System Officers Course at Keesler AFB, Mississippi. In August 1985, he was assigned to the Network Development and Integration Office at Sunnyvale AFS, California, where he directed the development of computer systems to support DOD space resources until September 1987. He was then transferred to the Network Control and Data Handling System Program Office at Los Angeles AFB, CA as a Range System Acquisition Engineer. In December 1989, he was chosen for the position as Chief, Data System Project Engineering Division where he supervised the acquisition efforts of 12 Air Force project engineers until entering the School of Systems and Logistics, Air Force Institute of Technology, in May 1991.

Permanent Address: 35 Wianno Road
Yarmouth Port, MA 02675

Vita

Steven C. Bishop was born on 26 December 1959 in Bakersfield, California. He graduated from Highland High School in Bakersfield in 1978, and enlisted in the Air Force in June 1980. After completing basic training and computer operator school, he was stationed with the 92nd Bombardment Wing (SAC) at Fairchild AFB, Washington. He worked as a computer operator in the Data Automation Division of the wing resource management deputate for three years, completing his Associate in Arts degree in his off-duty time. In 1983 he was selected for the Airman Education and Commissioning Program to complete his Bachelor of Science degree in Electrical Engineering at Arizona State University. Upon graduation in May 1986, he was assigned to Officer Training School at Lackland AFB and received a reserve commission in September 1986. He was then assigned to the engineering directorate of the Air Force Satellite Control Facility at Sunnyvale AFS, California where he managed communications engineering projects for command and control of DOD space assets. In 1987 he moved with the engineering directorate to HQ Space Division, Los Angeles AFB, and continued managing communications and computer system projects. In May 1991 he entered the School of Systems and Logistics at the Air Force Institute of Technology.

Permanent Address: 2912 Century Drive
Bakersfield, CA 93306

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1992	Master's Thesis

4. TITLE AND SUBTITLE
RANGE SCHEDULING AUTOMATION FOR THE AIR FORCE SATELLITE CONTROL NETWORK: A CASE STUDY IN COMPUTER SYSTEM DEVELOPMENT

5. FUNDING NUMBERS

6. AUTHOR(S)
Donald J. Aitken
Steven C. Bishop

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GSM/LSY/92S-1

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This study explored the factors influencing divergent outcomes of two computer system development efforts which were undertaken to fulfill the same requirement for computer automation of a manual resource scheduling process. The first project employed the traditional waterfall approach to system development, but resulted in user rejection and cancellation after considerable resources and effort had been expended. The second project employed prototyping and both the process and the product were well received by the users and ultimately produced an operational system. Analysis yielded eight contributory factors to the failure of the first effort. Three of these were related to the waterfall approach, but the remaining five would have adversely affected any type of development effort. As a result, the waterfall approach was not deemed to be the most significant contributor to the failure. However, the major contributor to the success of the second effort was the use of prototyping. Most theoretical advantages of prototyping over the waterfall approach were observed in that effort and two additional advantages were identified. Prototyping's disadvantages were largely mitigated by strong management control of the development process.

14. SUBJECT TERMS
Software Engineering, Prototypes, Requirements, Computer Programs, Methodology, Case Studies

15. NUMBER OF PAGES
164

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

AFIT Control Number AFIT/GSM/LSY/92S-1

AFIT RESEARCH ASSESSMENT

The purpose of this questionnaire is to determine the potential for current and future applications of AFIT thesis research. Please return completed questionnaires to: AFIT/LSC, Wright-Patterson AFB OH 45433-9905.

1. Did this research contribute to a current research project?

a. Yes b. No

2. Do you believe this research topic is significant enough that it would have been researched (or contracted) by your organization or another agency if AFIT had not researched it?

a. Yes b. No

3. The benefits of AFIT research can often be expressed by the equivalent value that your agency received by virtue of AFIT performing the research. Please estimate what this research would have cost in terms of manpower and/or dollars if it had been accomplished under contract or if it had been done in-house.

Man Years _____ $ _____

4. Often it is not possible to attach equivalent dollar values to research, although the results of the research may, in fact, be important. Whether or not you were able to establish an equivalent value for this research (3, above) what is your estimate of its significance?

a. Highly Significant b. Significant c. Slightly Significant d. Of No Significance

5. Comments

_____ _____

Name and Grade Organization

_____ _____

Position or Title Address