

AD-A045 274

SACRAMENTO AIR LOGISTICS CENTER MCCLELLAN AFB CALIF D--ETC F/G 9/2
SOFTWARE ENGINEERING TECHNIQUES IN COMPUTER SYSTEMS DEVELOPMENT--ETC(U)
DEC 76 J H LEHMAN

UNCLASSIFIED

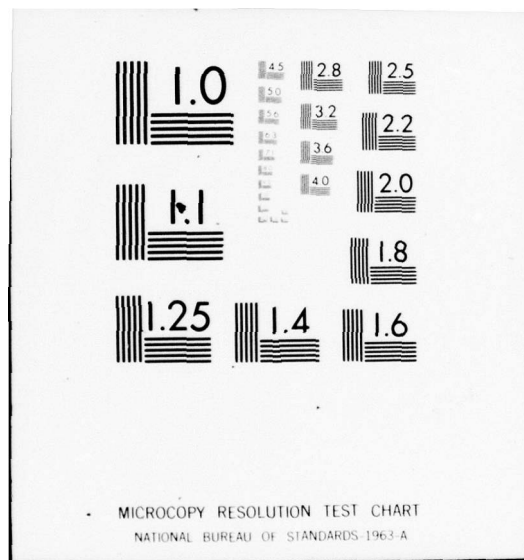
SM-ALC/ACD-76-04

NL

| OF |
ADA045274



END
DATE
FILMED
11-77
DDC



AD A 045274

Report No: SM-ALC/ACD-76-04
15 December 1976

1
B.S.



SOFTWARE ENGINEERING TECHNIQUES

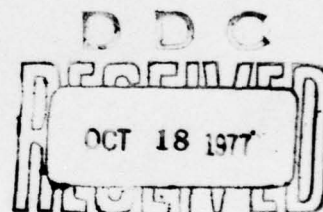
IN

COMPUTER SYSTEMS DEVELOPMENT

John H. Lehman

Lt. Colonel, USAF (Retired)

Distribution Statement A
Approved for Public Release;
Distribution Unlimited



AD No. —
PNC FILE COPY

DEPARTMENT OF THE AIR FORCE
HEADQUARTERS SACRAMENTO AIR LOGISTICS CENTER (AFLC)
McClellan Air Force Base, California 95652

ABSTAINER

This research report represents the views of the author and does not necessarily reflect the official views of the Data Automation Branch or the Department of the Air Force.

This document is the property of the United States Government and is not to be reproduced in whole or in part without permission of the Data Automation Branch, McClellan Air Force Base, California.

FORWARD

On 6 January 1976, the Data Automation Branch of Sacramento Air Logistics Center (ALC) initiated the development of a major logistics software capability, the "Allied Recoverable Requirements Computation System" (ARRCS). This was the first major software development task assigned to this ALC since 1973 and represented our first opportunity to employ the new software engineering techniques, (Top Down Design and Structured Programming) then coming into their own in the data processing community. This center had initially been exposed to "IBMs Modern Programmer Productivity Techniques" through a series of courses conducted early in 1974; however, because of lack of development workload, had been unable to implement these concepts.

It was determined that ARRCS would be developed using a combination of techniques drawn from:

1. The Rome Air Development Centers project reports on Structured Programming,
2. IBMs Modern Programmer Productivity Techniques, particularly the use of HIPO charts, and
3. the concepts expounded in Fred Brooks' book, The Mythical Man Month.

At the time that this new development was started, a determination was made that at least the initial activities would be thoroughly documented in an attempt to ascertain the value of the new software engineering techniques. To aid in this I directed a complete list of project records be kept, including a weekly diary, and a monthly review and report. Capt Daniel Wagner, the project officer, further required a periodic "dump" of all thoughts and ideas that his team had on this subject. These reports, other documentation, the programs themselves, and all correspondence, form the source material for this study.

I was fortunate to have assigned to the Data Automation Branch Lt Col John H. Lehman, shortly before his retirement, for he volunteered to research and write this report.

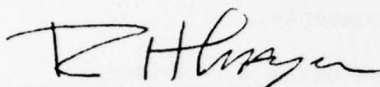
Col Lehman was not assigned to the Data Automation Branch until 5 Aug 76, after the completion of ARRCS phase 1, therefore, all of his views and conclusions were drawn from interviews and the source material. He worked fulltime on the report until his retirement on 30 Oct 76, and part-time until its completion in mid-December. Col Lehman is now retired, is a fulltime student, and working as a Data Processing Consultant.

ACCESSION for	
NTIS	Write Section ☒
DDC	Diff Section ☐
UNANNOUNCED	☐
JUSTIFICATION	☐
BY	
DISTRIBUTION/AVAILABILITY CODES	
Dist.	SPECIAL
A	

I believe this study accurately reflects "lessons learned" in this, our first attempt to employ the techniques of Top Down Design and Structured Programming.

Correct decisions and errors are reported with equal candor in the hope that the reader will benefit by our mistakes when implementing his first software engineering project. Many of Col Lehman's recommendations have been implemented in phase 3 of ARRCs and our organization is deeply indebted to him.

In addition to acknowledging the fine work done by Lt Col John H. Lehman, I would also like to thank the project leader, Capt Daniel L. Wagner, Mr Carl Distefano, senior mission analyst, Mrs Joyce Vittone, the project administrator, and every member of the ARRCs development team, both programmer and logistician.



RICHARD H. THAYER
Colonel, USAF

ABSTRACT

This case study centers on the effort involved in the development of a major logistics software system.

The development effort was unique in several major respects. It is divided into three general groupings. The first is historical and describes in the most general terms the system itself. The history predates the development effort by some months and terminates roughly with the certification of its initial phase.

The second part centers on a review and evaluation, not of the system, but of the procedures followed in its development. This evaluation is almost totally derived from commentaries, both written and oral, of the participants.

The final third of the text provides suggestions which the author hopes will prove useful in the development of the future phases of the System or totally new software development efforts undertaken at the Sacramento Air Logistics Center.

TABLE OF CONTENTS

SECTION	PAGE
Forward	iii
Abstract	v
Preface	ix
I HISTORY	
System Background	1
The Problem	1
The USAF Solution	2
The FMS Solution	3
Pre-Development History	4
Development History	5
II TECHNICAL REVIEW AND EVALUATION	
Manning-Personnel-Environment	11
General	11
Composite Profile	12
Programming/Analysis	13
Requirements System	14
Management	14
Environment	15
Physical Facilities	15
Organization	15
Documentation	16
General	16
The Functional Description (FD)	16
The Development Plan (DP)	17
The Data Element Directory (DED)	18
Software Engineering Techniques	18
The Development Team	18
Walk-Thru's	23
HIPOS	24
Development Support Library	25
Top Down Design	25
Top Down Implementation	25
Structured Programming	26
Major Handicaps	26
Training	27
Development Structure	28
System Definition	29
The Target Date	30
Statistical Recap	30

III RECOMMENDATIONS

General	31
Organization	
Applications Programming	31
Systems Programming	32
Training	33
Team Arrangements	33
Documentation	34
Software Engineering Techniques	
The Development Team	35
Walk-Thru's	36
Standards/Procedures	36
Administrative Procedures	37
System Definition/Target Dates	38
Abstainer	41

List of Illustrations:

FIGURE	PAGE
1 ARRCs Manning Chart	12
2 The Design Team	19
3 The Implementation Team	20
4 The Evolved Team	21
5 Sample Format	38

PREFACE

On the sixth of January, 1976, the development of the Allied Recoverable Requirements Computation System (ARRCS), a major application software capability was officially initiated at the Sacramento Air Logistics Center. This effort was unique in several major respects. The ultimate user or users of the ARRC System, those nations acquiring military aircraft from the United States, were not involved in any but the most remote way with its development. The myriad of potential users, and the limitless combinations of hardware upon which the system might possibly be employed dictated severe design constraints and tended to reinforce the premise that only through the application of the principles of top down design (TDD) and structured programming (SP) could a capability be developed with the requisite flexibility. TDD and SP were therefore selected to be the cornerstones of the development methodology employed in bringing this system to life.

What follows is an analysis of this effort. I have divided the study into three general groupings. The first is historical and describes in the most general terms the system itself. The history will predate the present development effort by some months and terminate roughly with the certification of Phase I ARRC by a team of logisticians from Headquarters, Air Force Logistics Command.

The second part of the analysis will center upon a review and evaluation, not of the system, but of the procedures followed in its development. This evaluation is almost totally derived from commentaries, both written and oral, of the participants. As might be expected, not everyone involved shared the same opinions and, where a wide divergence existed, greater reliance was placed upon the statements of those closest to the area in question, rather than the most or least critical or laudatory.

The final third of the text will provide suggestions which the author hopes would prove useful in the development of the further phases of the ARRC System or totally new software development efforts undertaken at the Sacramento Air Logistics Center. As in the case of the preceding paragraph, the majority of the inputs originated with the project participants. In final form most suggestions are an amalgam of several proposals elaborated upon by the author.

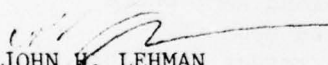
This study differs in several major respects from similar studies of top down design and structured programming efforts which have come to my attention. First, the author was not involved in the development of the system itself but rather became associated with it only after its Phase I completion and then only for the purpose of the study. Second, it was to be a history of the total development effort and not just its technical aspects, and finally,

recommendations were to be provided covering a broad spectrum ranging from technical detail through organization.

The research pattern followed entailed: 1) Reading all available files and documents, identifying and cataloging those items deemed to be significant; 2) Establishing a tentative framework (outline) for the study; 3) Relating documents to the proposed outline and determining their adequacy in furnishing the requisite data; 4) Developing a questionnaire (to be used in interviewing each participant) covering in large measure those gaps that appeared in matching the available documentation to the proposed outline and finally; 5) Interviewing each available participant.

As a final note, I believe it necessary that I establish what credentials I have to qualify me to perform this service. I have been closely involved in Data Automation for eight of the last ten years. An initial two years was spent in the maintenance of the SAGE (Semi-Automated Ground Environment) System. During six of the following eight years, I functioned as super analyst programmer for several medium to large application software capabilities, was Chief of the Command Control Programming Branch, Chief of the Plans and Requirements Division, and had a stint as Deputy Director of Data Automation; all these at Headquarters PACAF (the Pacific Air Forces). At the time of my departure from PACAF, the Analysis and Programming Division had just initiated its first effort at TDD and SP, so my practical experience in this area is limited.

My thanks to all the members of the development team whose complete support and candor made this study possible. Also, my special thanks to my secretary, Ms Gindi Flemming, for her typing and redaction of this technical report.


JOHN H. LEHMAN
8189 Plumeria Avenue
Fair Oaks, California 95628
(916) 966-5189

SECTION I

HISTORY

System Background

The Problem.

With the ever increasing sophistication of military hardware, modern aircraft being a prime example, there has been a concurrent increase in equipment costs. As would be suspected, not only has original purchase price risen in terms of real dollars, but the repair and replacement costs of individual components has also been subjected to sharp increases. With some repairable/replacement items costing in the hundreds of thousands of dollars few nations, if any, could afford an arbitrary buy of six or twelve of each on the off chance that that quantity would ultimately be used or even be sufficient to the task.

The expense of reestablishing a production line for a small run could add astronomically to the cost of an item, whereas failure to do so when the part is needed, could result in the effective loss of each aircraft as that component became damaged beyond repair. On the opposite end of the spectrum is the overbuy which could result in an inventory of unique parts, all purchased at great expense, and all reduced to the value of scrap when its associated weapons system was phased out.

The problem is one of having just the proper amount on hand at any one time to meet immediate and crisis needs while also forecasting requirements far in advance or through the life cycle of the weapons system. The factors that must go into such a calculation are manifold. Representative of the considerations involved are: the flying hours programmed per aircraft per period of time; failure rates on specific items; the status of failed items, be they locally repairable, depot repairable, or condemned (beyond economical repair); enroute times to maintenance facilities; and repair cycle times once the defective part has reached the depot.

Each country or geographical area has unique characteristics that can alter the above considerations. Flying an aircraft 100 hours a month, will surely result in a greater number of component part failures than flying only twenty. The quality of flight and ground crews will have a large effect, as will environmental conditions. Blowing sand may cause rapid failure of certain components whereas extreme cold will have a detrimental effect on others. Though all this data could be manipulated by a single automated system with a

single data base projecting worldwide requirements, each customer receiving a pro rata share of de-icer boots, as an example, would prove of marginal utility at best.

Foreign Military Sales (FMS), that is the transfer of U.S. Military and Related Hardware to foreign countries, are conducted on a cash, loan, or grant basis. Assuming that it is in the best interest of the United States that maximum effective use of this equipment be made and maximum benefits be derived per dollar spent, the above delineated problem requires a solution. Beyond that, since each recipient of U.S. Military Hardware has unique requirements, any solution must take these variations into consideration. A system to accomplish this has been called a Recoverable Requirements Computation System.

The U.S. Air Force Solution

As might be expected, the United States Air Force has such a system already in existence. The problem described above, or rather its solution, is therefore simplified since: 1) Methods of calculation employed by one system may be employed by another; 2) A body of experience exists in the development and employment of such a capability; 3) Data is available, arrived at empirically, as in the case of consumption factors, or analytically, as in the determination of what constitutes a repairable asset.

Three possibilities come immediately to mind. Why not use this capability on behalf of each FMS recipient? Or, why not provide each FMS recipient with a copy of the software for their own use? Or, why not convert the present system to operate on FMS recipient hardware? There are many objections to each of the above courses of action and I will only state the basic reason each was ruled out as a viable alternative.

In processing/calculating individual FMS requirements at a continental United States facility, the system would be almost totally unresponsive to the local manager responsible for making buy decisions. Data input and product output would be hopelessly behind the times. With a possible 63 FMS countries requiring this service, the administrative problems, not to mention computer hardware requirements, would be horrendous.

To provide each FMS recipient with present programs would be futile. The present system, actually a number of sub-systems, is operated largely on a second generation IBM 7080 computer. Most FMS recipients are just now entering into the computer era and, quite naturally, purchasing or leasing third rather than second generation equipment. In most instances, conversion and integration of present USAF software would be beyond the capabilities of indigenous programming staffs in FMS recipient countries.

To further expand on the theme of conversion, an inhouse analysis of the efficacy of a direct conversion of the present Air Force System resulted in the conclusion that a total re-write incorporating already identified requirements and enhancement would be the preferable course of action. Among the reasons cited, were the state of present system documentation and the large number of modifications already made that have naturally resulted in a less than straightforward, easily followed, program code.

The Foreign Military Sales (FMS) Solution.

Since employment of the USAF system(s) either directly or following a conversion was judged to be a less than adequate alternative, the only remaining choices were to develop a totally new system or develop no system at all. In choosing to create ARRCs for a wide and, to a great extent, unknown constituency the developers took on a burden that would not accrue to a more conventional development activity.

The constraints and considerations under which the design evolved included the following. The system must:

1. with only minor modification (to facilitate installation) be capable of operating on any third generation computer that had available an ANS COBOL and ANS FORTRAN compiler.
2. be tape oriented, thereby dictating sequential processing.
3. have the capability to accept a variety of input formats with little system modification to allow interface with existing capabilities.
4. be adaptable to special requirements of individual users.
5. be simple to operate with extensive data edit capabilities.
6. be modular to facilitate
 - a. implementation of portions of the system if the full capability is not required.
 - b. system enhancement to provide more sophisticated data handling and manipulation as required.

Pre-Development History

Though the necessity of establishing methods and procedures for determining, with accuracy, recoverable item requirements of nations engaged in purchasing (obtaining) U.S. Military Hardware has been under discussion for many years, the earliest serious reference to the development of a computer capability for this purpose that the author could find was dated February 1975 when, during the course of a VNAF (Republic of Vietnam, Air Force) Requirements Computation Conference, the Sacramento Air Logistics Center, Directorate of Material Management, SM-ALC/MMM, agreed to prepare specifications for a very simplified automated requirements computation system. The Air Force Logistics Command, Directorate of Data Automation (AFLC/AD) was to write, test, and operate the developed system. By early March '75, AFLC had reconsidered its role as system developer and this now devolved upon SM-ALC. Further, on 10 March 1975, AFLC/AD directed SM-ALC/MM to prepare the Data Automation Requirement (DAR) and System Specification (SS), by April, at which time SM-ALC/AC (Office of the Comptroller) was to review the completed documentation and assess its capability to support the development workload. By the third of April, it became evident that our client state would have no further use of a Recoverable Requirements Computation System and authority to expend DAV funds for continued development was cancelled. On 9 April 1975, however, HQ AFLC requested that the DAR and specification be completed by "18 April 1975 as originally planned", citing the real and valid need for the system by other Security Assistance Program countries.

The DAR, titled Allied Recoverable Requirements Computation System (ARRCS), was completed on 22 May 1975. Development costs in terms of manpower were estimated to be 66 man months of computer programmer/analyst time and 24 man months of effort on the part of logistics systems analyst. In forwarding the completed DAR to HQ/AFLC, SM-ALC/MMM indicated that the functional description (FD) and Systems Specification (SS) would be provided upon completion. A review of the available files in ACDA shows no further activity until 18 July 1975, when SM-ALC/MMML (Advanced Logistics Systems Branch) at the instigation of HQ/AFLC/MMI (Directorate of International Logistics, Office of DCS/Materiel Management) requested SM-ALC/AC provide inputs to derive an estimated cost to design, develop, and implement the ARRC System for the Iranian Air Force without producing all the documentation required by the Department of Defense Automated Data System Documentation Standards Manual, DODM 4120.17-M.

The estimated cost of the system, not taking into account the Iran unique expenses, (TDY, Training, et al) was decreased from \$146,790 (reflected in the ARRCs DAR) to \$67,343, representing a drop of just over 54%. I hasten to add that the system sketched in the DAR was considerably smaller and far less sophisticated than the system finally approved for development, and since added to and embroidered upon.

On 29 August 1975, SM-ALC/MM was given authority to continue with the development of the ARRC System "until completion of the current phase", this being the development of a Functional Description and System Specification. It was thought likely at that time that the system would be employed in Iran under Project Peace Log and on 5 September a message from AFLC confirmed that billing would indeed be against that project. It stated further that "4120.17-M documentation does not apply" and requested a completion date for the System Description and Specifications. On 9 September 1975, AFLC was advised that a target date of 15 October 1975 had been established for completion of the requested documents and suggested a review of the completed work be conducted at SM-ALC by AFLC/MMI/ACD during the week of 16 October. On 7 October the incorporation of "Variable Safety Level" into the ARRC System was requested by AFLC/MMI and concurred in by SM-ALC/MMM. A new review date of 3-7 November was also agreed upon.

On 3 November, when the review was conducted, the only document available was a draft of the System Specification. On 21 November, SM-ALC/ACD was notified by AFLC/MMI that funding for the system would no longer be provided through Project Peace Log. Further, SM-ALC/ACD was requested to send someone to HQ/AFLC to discuss ARRC development for initial implementation in Saudi Arabia under the sponsorship of Project Peace Hawk. On 25 November 1975, the System Description prepared by SM-ALC/MMM, and dated 6 November 1975, was received by SM-ALC/ACD.

The Directorate of International Logistics, Office of DCS/Materiel Management advised SM-ALC/MMM/AC on 23 December, that, "The current requirement and authorization for SM-ALC to develop the Allied Recoverable Requirements Computation System under Project Peace Log, has been changed to Project Peace Hawk for further funding." The letter went on to state in paragraph 2, "Under this new agreement, the specifications for the ARRC System will call for the addition of mathematical models, expansion of documentations requirements, and various other changes which have been discussed with your ACD personnel." The authorization in paragraph 1 was followed by a request in paragraph 3, that AC and MM accept the development responsibility, identify project officers, prepare a milestone chart, and justify resources required.

With SM-ALC/AC/MM acceptance of the expanded development effort on 6 January 1976, the project was officially launched.

Development History

Though a considerable amount of time and effort had already been expended on the ARRC project, and though the official start date must be 6 Jan 76, the serious effort dates from 23 December

1975 and the request made by HQ AFLC/MMI that SM-ALC/MM/AC undertake this task as a joint venture. By the sixth of January a milestone chart had been prepared showing a four phased development effort with the first target event, [system] definition, to be completed in mid-January '76, and the last target event, [in-country] implementation, forecast for completion by the end of August 1977. The implementation of phase 1 "Basic D041" was targeted for mid-June '76.

Manpower requirements had also been determined to be five Mission Analysts (employment, as opposed to computer analysts - provided by the Logistics Systems Management Division), eleven computer specialists, one clerk-typist, and, parenthetically, one supervisory computer analyst. The development team grew at a more or less constant rate until it reached its full compliment in March of 1976. There was no turnover of programmer/analysts until close to the end of Phase 1 development when one change was made, though several changes and substitutions were made in the ranks of the Mission Analysts.

The major task confronting the ARRCs team for the next two months was the development of the functional description (FD). Since the "expansion of documentations requirements," in the letter of authorization apparently alluded to DODM 4120.17-M standards, any savings that were to be realized by minimizing the amount of documentation produced were now eliminated. The cost of producing good documentation however, is often more than offset in the system's maintainability, utility, and longevity.

In the letter of 6 Jan 76 accepting development responsibility, it was stated that "a system requirements review should be accomplished the week of 2 Feb 1976" and this conference was ultimately held at SM-ALC during the week 18-25 February 1976. In attendance were representatives from SM-ALC, AFLC, OO-ALC (Ogden Air Logistics Center) representing the F-16 System Manager, and Northrop Worldwide Aircraft Services, Lawton, OK (NWASI). NWASI was involved due to negotiations concerning contractual support to be provided Saudi Arabia in programming and operating its Aircraft Replacement Components Logistics/Supply System. In preparation for this, NWASI had an IBM 370/135 computer alleged to be in the same configuration as a machine to be installed in Saudi Arabia at some date in the then not too distant future.

To paraphrase a talking paper prepared by SM-ALC/ACDA: The purpose of the meeting was to review the ARRCs functional description, thus assuring that the user's needs had been thoroughly identified and were clearly stated. In addition, the development plan would be reviewed to insure that ARRCs was developed and documented in an orderly manner using the new techniques. The aim of the conference was to establish a functional base line and firm commitments regarding documentation and manning of the project. It was essential that all parties agree on the scope of the work. AFLC/MMR had agreed to support this project because they saw a potential use by AFLC.

During the course of the review, AFLC directed some changes and additions be made to the draft functional description. In a message dated 3 March 1976, AFLC/MI requested that two ARRCs development team personnel travel to HQ AFLC for the purpose of reviewing and discussing the revised FD and developing a command ARRCs briefing. Travel was accomplished between 7-10 March. In addition to the FD review/approval and development of the command briefing, the meeting produced an item of major significance. While the due date for Phase I (the first increment) was confirmed as 20 June 1976, the completion date for the final increment, (Phase IV) was moved ahead one year and four months to 1 January 1979. Reasons for this change in final implementation date were given as:

1. More stringent error detection and correction parameters.
2. Stock levels computed individually by base rather than employing depot averaging techniques.
3. Each package (phase/increment) to be developed and demonstrated as a stand-alone system.

By mid-March the functional description (FD) and development plan (DP) had been completed and approved.

While the functional description and development plan were being placed in final form an in-house training effort was initiated to acquaint or reacquaint the programmers/analysts on the development team with the principles and techniques of top down design and structured programming. Classes were held for two hours a day, for eight consecutive duty days starting on the third of March. At the end of this period, it was estimated that six additional hours of instruction would be required. This was tentatively scheduled for the week of 22 March, but never materialized.

During that period a solution had to be provided for the problem posed by the non-availability of a computer of the target machine type and configuration within the development center. There were several iterations on how to solve this dilemma. The DP Solution entailed initial development on the CDC Cyber 70, followed by conversion and test on the IBM 360/44, operated by the 1155th Technical Operations Squadron at McClellan AFB, with the final adaptation to an IBM 370 being accomplished at Lawton, OK in the facilities of Northrup Worldwide Aircraft Services, Inc. (NWASI).

Another iteration, also not acted upon, considered the acquisition of a 370/115 through NWASI for installation at McClellan AFB.

The final solution provided for the lease of IBM 370 computer time from a private firm, California Health Data Corporation (CHDC), and, though a certain amount of inconvenience was experienced, and a great amount of time and effort expended by one team member, the administrator, in accomplishing this, the overall effect was positive. The initial face to face contact with CHDC was made on the sixteenth of March. By 12 April agreement was reached on costs and services and on the fifteenth of that month, the contract was signed and went into effect. Provisions included the designation of a work-area, guaranteed processing time, and presentation of a three hour training session for ARRCs programmers on 370 DOS/VS. The period of the contract extended through 31 July 1976, at a net cost of \$23,413.68.

On 20 April, the California Health Data Corporation conducted the contracted for 370 DOS/VS training and during the following week, the initial compilation of ARRCs programs was undertaken.

On the ninth of April, notification was received that the ARRC System demonstration previously planned to take place in Saudi Arabia was now to be accomplished at the NWASI facility, Lawton, OK. The reason given was the failure of the Saudi Arabian government and Northrup Worldwide Aircraft Services, Inc., to finalize negotiations.

By the third week in May, three members of the ARRCs development team went to Lawton, OK to test pieces of the ARRC System in the NWASI environment and obtain first-hand knowledge of the system generation and operational configuration of that computer.

The operating system software employed at NWASI was not in any way compatible with the DOS/VS system employed at CHDC. Before extensive testing could be accomplished on the ARRCs applications software at that facility, a complete system generation would have to be provided by the ARRCs team and the system initial program loaded (IPL's). To implement ARRCs in this environment would require stand-alone operation and necessitate an IPL each time processing was initiated. The development of a system generation (IPL) became the first order of business following the acquisition of system engineering support from CHDC. The requirement for operating system support in the form of a systems engineer was recognized early in the development effort, the earliest written reference being dated the twenty-second of March. Unfortunately, contract negotiations between IBM and the Air Force to acquire this support dragged on over the issues of "liability" and "guarantee" (which IBM would not accept or extend). On the twenty-fifth of May, a contract was awarded to CHDC to provide this service.

From the standpoint of familiarization with the NWASI environment, the trip proved a success but pointed up the requirement once more for an IBM systems engineer (operating systems programmer) to be available to the ARRCs development team.

One further element caused concern and that was the lack of understanding on the part of both NWASI and SM-ALC about what their opposite number was developing. It all became moot shortly thereafter, however, when NWASI dropped out of the equation entirely. Though several trips to Lawton, OK had been planned, including one for the demonstration of Phase I ARRCs, only the single TDY between the sixteenth and the twentieth of May was ever conducted. With NWASI out of the picture the demonstration locale was moved to CHDC, with the tentative date remaining unchanged.

Programming and testing continued through May and June and though no figures are available, the frequency and length of walk-thru's declined as the required delivery date approached.

In mid-June, a meeting was held at Wright-Patterson Air Force Base for the purpose of briefing cognizant individuals on the progress of ARRCs development and "firm up the demonstration date". Broadening of the ARRC System was also to be discussed. Both SM-ALC/ACD and MMM were in attendance. The immediate outcome of the discussions was the establishment of 19-21 July as the demonstration time frame for Phase I.

In early May, interest was shown in the acquisition of a "Librarian Software Package" to aid in the development effort. Among other things, the librarian software would facilitate program testing, provide an audit trail of program changes, allow the inclusion of temporary modifications, and produce summary output products showing lines coded, contents of program libraries, etc. On the twenty-third of June, the "Librarian" purchased from Applied Data Research, Inc. was installed at the CHDC facility but not without some problems that resulted in the loss of two days of program test time.

In early June, a "training plan" was completed and defined requirements ranging from Introduction to System 360 Programming through Training for Project Management. On the eighteenth of June, approval was received from HQ AFLC to carry over funds from 76/4 to 77/T to be spent for training in support of the ARRCs project.

For the week ending 28 May, the activity report read:

The computation group has tested the OIM segment; the input group has some conversion, F/M [File Maintenance] and edit modules tested, and the output group is coding the June demo products. The output group may require overtime to complete the June demo.

The week ending 18 June:

The computation run has run to completion and is approximately 98% tested. The output worksheet has been tested for pages 1 and 5 and part of page 3.

Input conversion is approximately 95% tested.

On the ninth of July, the full five page computation worksheet was prepared and given to the Mission Analysts to research. A total of 28 discrepancies were identified by MM and AC personnel. During the week ending the tenth of July, testing and debugging continued and problems were compounded by a power failure at the CHDC computer installation late in the week and both Saturday and Sunday, (the seventeenth and eighteenth of July) were used to put the finishing touches on the system.

On the twenty-second of July, Phase I of ARKCS received certification from a team of logisticians sent by the Air Force Logistics Command for that purpose. Some minor discrepancies were noted.

SECTION II

TECHNICAL REVIEW & EVALUATION

Manning-Personnel-Environment

General.

Development of the Allied Recoverable Requirements Computation Systems (ARRCS) application software and its attendant documentation was a joint venture of the Sacramento Air Logistics Center (ALC) Data Automation Branch, Office of the Comptroller and the Logistics Systems Management Division of the Directorate of Material Management. Though authority to proceed with the development was received in late December of 1975, some preliminary work had, of necessity, already been initiated. From 24 November 1975 through March 1976, the number of programmer/analysts involved grew from one to 14 at a more or less constant rate and with only one exception occurring in early July there were no terminations or transfers experienced prior to the completion of the initial development effort documented in this study. In aggregate approximately 60% of one programmer's time was spent on maintenance of other capabilities and of this amount, the bulk was employed in support of the DO41A system, the logical stepfather of the ARRCS. A secretary was assigned in mid-January and has also provided support for the ARRCS project without significant interruption.

Initial planning called for three full time analysts to be provided by the Logistics Systems Managements Division. However, this was later raised to five. Between mid-January and mid-July, this number averaged between five and six, with only one brief period in February showing only four assigned. As can be seen in Figure 1, a total of ten individuals were used to fill the five Mission Analyst positions over the period of Phase I, ARRCS Development. In order to differentiate between Computer Programmer/Analysts and Logistics Systems Analysts, the latter will be referred to as Mission Analysts for the remainder of this report.

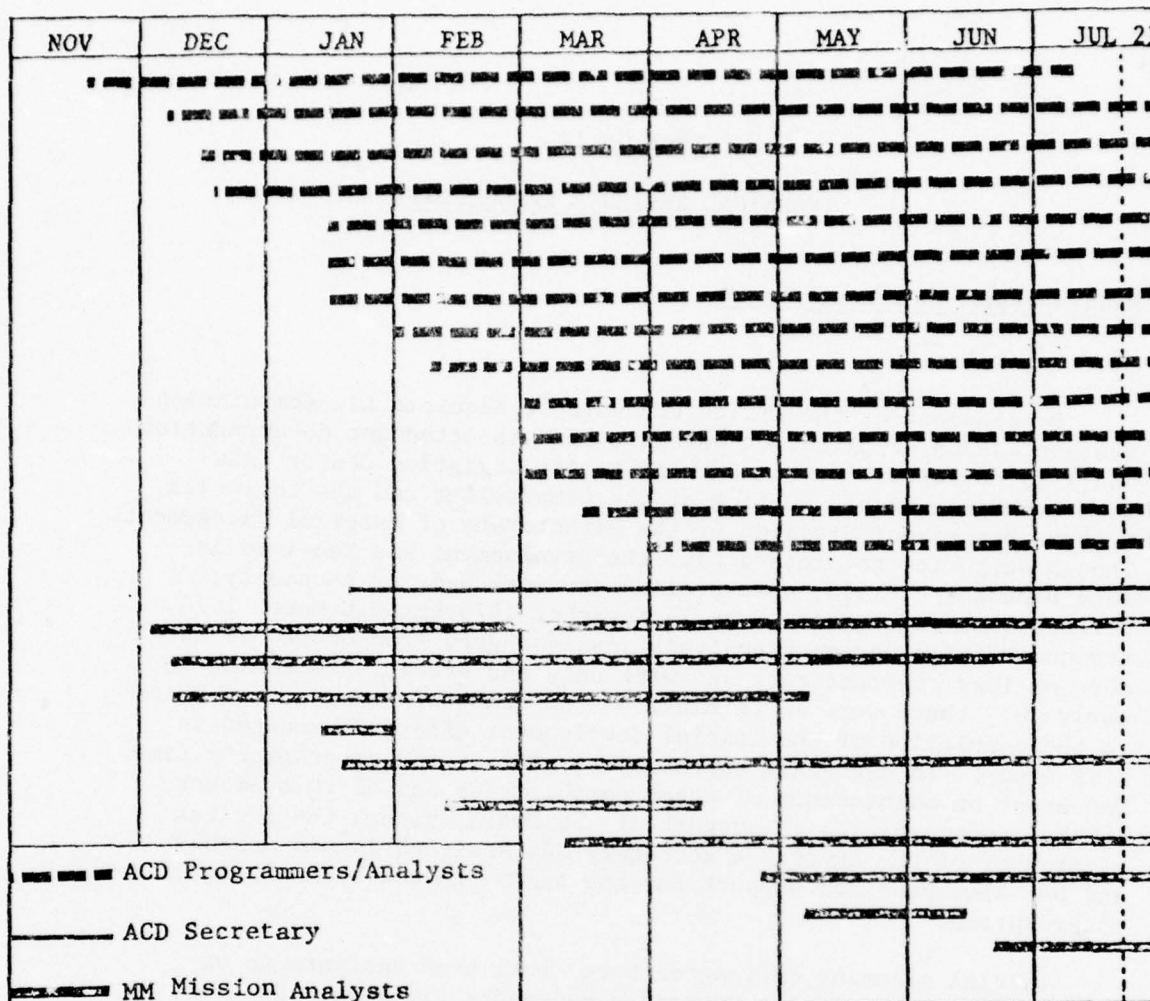


Figure 1. ARRCs Manning Chart

Composite Profile.

In developing a composite profile, three areas of experience or expertise were considered. These were 1) Computer Programming/Analysis; 2) Requirements Systems and; 3) Management.

Under the general heading of Computer Programming and Analysis are include experience with COBOL Programming, Third Generation Hardware, Job Control Language, Top Down Design and Structured Programming, and Original System Design and Development (as opposed to System Maintenance or Add-Ons).

Computer Programming/Analysis

Cobol Programming: A straight mathematical tally of COBOL programming experience for the fourteen programmers involved, shows an impressive seven year average. High time was 22 years and low was six months.

Third Generation Hardware: Experience on third generation hardware ranged from zero to just over seven years. Average experience approximated three years with the bulk of that having been acquired on the CYBER 70. A total of ten years experience was on IBM 360/dd equipment with no experience on the IBM 370/ddd, the target hardware for the ARRC System.

Job Control Language: Through JCL in one form or another is characteristically employed on most third generation computers there is not a direct correlation between JCL and third generation equipment experience. JCL for the Burroughs B-3500 is limited and there is no format compatibility between CYBER 70 and IBM 360-370 control language. Beyond that, the depth of experience can vary greatly from a development activity where JCL creation must be done in total by each programmer through a maintenance function where Procedures (Proc Libraries) have been established and programmers operate with only a small, predefined, subset of the total JCL package.

Control Language experience for the team averaged approximately two and one third years, with an IBM JCL background being shared by only four members of the group, representing a total of ten and one half years.

Top Down Design (TDD) and Structured Programming (SP): Though three team members had attended a formal course of instruction in 1974, they acquired no measurable practical experience other than an application of some SP concepts to system maintenance and enhancement activities. One team member, without benefit of formal training in TDD had been applying the principles in some measure to development tasks.

Original System Development: Assessing experience levels in the development of original systems is an iffy thing at best. Sub-systems of a large capability may call for greater originality than the "from scratch" development of a small or even medium sized system. For that reason, development of major units of the Advanced Logistics System (ALS) were considered as meeting the criteria. Just what is a small or medium or large system is also not that easy to define. Programmers interviewed were given the latitude to determine for themselves which systems fell into what category though the suggestion was made that small systems were probably in the area of ten programs, medium systems in the vicinity of fifty and large systems circa 100 or better. With these firm guidelines established, it was determined that only three of the 14 Programmer/

Analysts interviewed had not engaged in the design of some unique computer software capability. A total of eight small, eleven medium, and four large systems (of which two were sub-systems of ALS) were developed. In only one instance did coding not accompany the design; but in only six cases, involving two individuals, did the development activities entail a level of supervision or personnel management.

Final Note: Though only Computer Programmer/Analysts were considered in this portion of the composite profile, one Mission Analyst had an extensive programming background having employed JOVIAL, ALGOL, FORTRAN, PL1 and SIMSCRIPT. Third generation hardware experience included the IBM 360 and HIS 6000. He had worked on the design and coding of six small and three medium sized systems. None of these tasks involved supervision.

Requirements Systems.

The availability of a body of prior experience with requirements systems was important for two reasons. The first centered on system complexity. Stated simply, unlike a standard supply system which deals with inventory, item issue, predetermined order points, special requisitions, etc., operating almost exclusively in the present, a requirements system must work in the past in an attempt to establish usage rates, the present to correlate inventory with requirements, and the future to extrapolate (or prognosticate) requirements over X months or years for the life of a weapons system. A Repairable Requirements Computation System such as ARRCs is greatly complicated by including not only replacement requirements as a straight new buy but also taking into account the repair or condemnation of recoverable items. Myriad factors as diverse as shipping times and depot repair cycles, must be considered in arriving at a time phased forecast uniquely tailored to each recoverable item in the weapons system.

The second reason that a Requirements System background was of importance was the lack of comprehensive, detailed, system specifications.

Of 14 Programmer/Analysts, three had prior Requirements Projection System experience. However, in one case that association had ended seven years earlier and in another, there had been a lapse of 22 years. The third Programmer/Analyst had a very extensive and current background, having completed six years in that area of the ALS Development Effort.

Management.

The final area of concern in developing the composite team profile was that of management experience. Here, because of its rumored uniqueness, we limited ourselves to prior experience in

software development. Of the 18 participants, only two had prior management experience. One supervised the activities of one to five programmers during the conduct of five unique development projects. The second had management responsibility for 15 to 16 programmer analysts in the development of a major sub-system of ALS and subsequently, for 150 individuals engaged in ALS system integration.

Environment

Physical Facilities.

Both Mission Analysts and Computer Programmer/Analysts shared a common facility at one end of a large room that comprised the ACD Management Section. Partitions were provided to allow some measure of privacy. Two small communal working areas with chalk boards and facilities for seating eight in cramped discomfort were carved out of a corner of the corner. Allocating each individual a proportionate share of walkways, keypunch, and communal areas, etc., floor space probably averaged 100 square feet per soul. An IBM 029 keypunch was available for programmer use. More remarkably, three pieces of equipment were made available for the exclusive use of the ARRCs Development Team: an IBM Mag Card II typewriter, acquired in January 1976, a Xerox copier, acquired 7 May 1976, and a Dictaphone system, incorporating telephone dictating devices at four locations, as well as two portable hand carried cassette recorders, acquired 30 April 1976. Many computers are in place at the installation, but of these, only a CYBER 70, a dual configured Burroughs 3500, and an IBM 360/40 are third generation machines. Further, the IBM 360/40 is dedicated to a single application and not available for unique development. An IBM 360/44 operated by the 1155 Technical Operations Squadron, Headquarters Command, located elsewhere on McClellan AFB can be made available on a lease basis for development activities during non-prime time.

Organization.

The Data Automation Branch (ACD) is organizationally directly beneath the office of the Comptroller (AC), Sacramento Air Logistics Center. ACD comprises four sections designated: Management, Base Systems, Logistic Systems, and Computer Operations. Each section, with the exception of Computer Operations, has as a constituent, a development group. Responsibility for ARRCs development was vested in the "Special for FMS (Foreign Military Sales) Development Group" created largely for this purpose and nominally placed under the Management Section. Mission Analysts are attached, rather than assigned, to the development group and retain their Materiel Management organizational identity.

Documentation

General.

There are three major documents that fit this category: the basic functional description and all but one of its appendices, which I treat separately, the development plan (normally section 6 of the FD but in this case developed as an entity, and the list of data elements which, though appendix 8 to the FD, qualifies as a document in its own right on size alone. Though DODM 4120.17-M served to provide the major framework, many of the formats originating in AFLC supplement 1 to that manual were also employed in preparing the text. Each document, for good or ill, had a pronounced effect on the further activities of the development team and should, in that light, be subject to further discussion.

The Functional Description (FD).

Development of the FD was a joint undertaking of SM-ALC/AC/MM. The purposes of this document as stated in DODM 4120.17-M were met to the extent possible. Those areas where compliance could not be achieved centered upon the uncertainty of, or lack of data on, ultimate system users. The FD did serve as a "basis for mutual understanding between the [proxy] user [AFLC/MI] and the developer." It provided information on performance requirements, preliminary design, and fixed costs, and could serve as "a basis for the development of system tests."

Appendices were employed where extensive data required in special paragraphs would have destroyed the continuity or readability of the document. Every item delineated under the rubric "this paragraph shall" was not provided, however, the FD was a remarkable piece of work, the more so considering the pressures and time constraints under which it was developed.

There can be no doubt that it received serious attention outside the development team, in its draft form at the Requirements Review, and subsequently in its final form by HQ AFLC/MI, at the time of its approval. A detailed review was also accomplished at HQ AFLC and produced two papers, one titled "Comments on Functional Description for Allied Recoverable Requirements Computation System" dealing primarily with changes to some of the formulas used in output calculations, and a second, "J.M. Hill Working Note #89" suggesting that the major product output by the Phase I system, the Detailed Requirements Computation Worksheet, be produced as two separate products, one dealing with repair and the other with buy requirements.

The major infirmity with the FD is that it was, in most respects, published in "final form" with little discernible effort made through the course of Phase I development to keep it current. No machinery was established to facilitate update though in some instances,

formula changes were posted by pen and ink and initial page-for-page revision of the data elements (Appendix 8) was attempted.

Without updating, the FD lost much of its value. A month after its publication, the statement "The functional description is not perfect..." and the exhortation "We must all work together to make it perfect" were made. A week later, a memo stated, "The FD as written is no longer a source document for resolving misunderstandings in the following areas:" It went on to list six technical questions and voiced a complaint about the data elements listing. However, the only formal revisions to appear were the aforementioned changes to Appendix 8.

The Development Plan (DP).

Though, as in the case of the basic FD, DODM 4120.17-M provided guidance in preparation of the DP, AFLC Supplement 1 to that manual was also relied on to some minor extent in deriving formats. Beyond that, a text entitled "The Mythical Man Month" appears to have figured prominently in defining the Software Engineering Techniques (DP Section 3) and the Structure of the ARRCs Development Team (DP Section 4). Evaluation of the DP is a difficult task. In many respects it went far beyond that which is required by the DOD manual. Since the AFLC supplement had been intended to provide additional guidance for the Advanced Logistics System documentation effort, the ARRCs team was under no obligation to comply with its provisions. Yet, the stated purposes of the ARRCs DP were a paraphrase of the purposes outlined in that document. Of the three objectives, only one and one half were met. Those related to documentation and the overall management approach. The others, "Pertinent Information on Resource Requirements", and "Schedule and Rationale for the Project Development and Implementation" were not addressed in any easily identifiable form.

Software engineering techniques encompassed the seven tenets associated with, and embracing, top down design and structured programming. These being: The Team Concept, Walk-Thru, HIPOS, the Development Support Library, Top Down Design, Top Down Implementation, and Structured Programming. The ARRCs team structure defined in the FD, was derived in large measure from the "Surgical Team" described in the Mythical Man Month. There were to be two teams, the first evolving into the second as the transition was made from design to design implementation. Highly stylized names such as Design Assistants, Tool Smith, Co-Pilot, and Language Lawyer, were used to identify positions or perhaps functions on the teams.

Other areas addressed in the DP were: Training (Sec 5), Collection of Historical Data (Sec 6), Status Reporting (Sec 7), Technical Report (Sec 8), System Design Phase (Sec 9), Programming Phase (Sec

10), the Gathering and Conversion of Data (Sec 11), the Program Test/Certification Plan (Sec 12), and Computer Hardware (Sec 13). All of these were in some measure directive in nature. However, most would have required, to one degree or another, implementing directives expanding upon the basic theme and providing in-house procedures before they could prove effective.

Data Element Directory (DED).

A data element directory (DED) was produced in an effort to assure the use of standard data elements in the programming activities. Two hundred thirty items were identified and described under the headings: Data Element Name, Mnemonic Name, Type/Length, Code Value, Source, Output, Function, and Description. Though Mnemonic names ranged from two to 30 characters, the number of long names exceeded the short by several orders of magnitude.

The DED went through several iterations. At first it was appendix 8 to the FD, and as such, was revised once. However, the volatility of this list resulted in its eventual "automation" in an abbreviated form. As data elements were added, changed, or deleted, new listings were produced and circulated. Unfortunately there was no way of identifying modifications without doing a line-by-line, word-by-word comparison with the previous list. Data division "copy" (include) members were not employed so enforcing the use of standard data element mnemonics was difficult at best. Dissatisfaction with the content of the DED as well as the length and propriety of the mnemonic names was a source of irritation throughout the development effort. Employment of the DED was honored more in the breach than in the acceptance.

Software Engineering Techniques

The Development Team.

As reported earlier, the DP defined two distinct team structures, one to be employed during system design, and the other to come into play with system implementation.

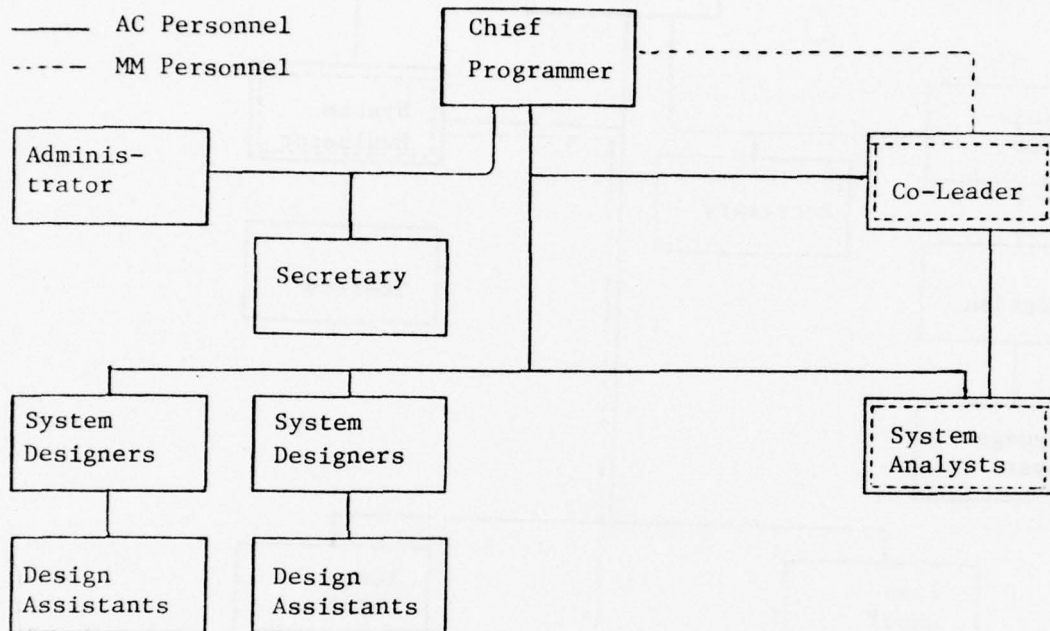


Figure 2. Design Team

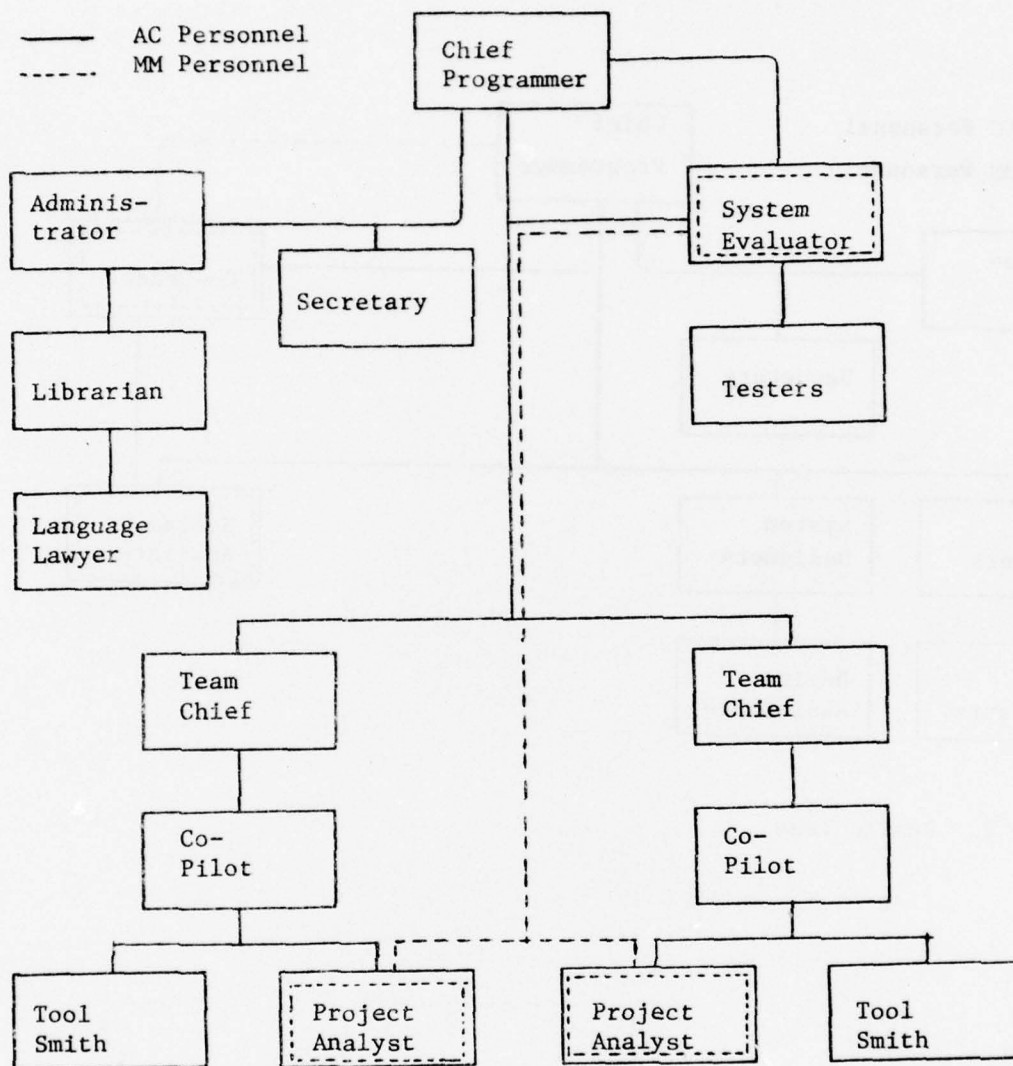


Figure 3. Implementation Team

A direct transformation from the design to the implementation team structure never took place and most individuals questioned indicated that they functioned in substantially the same manner throughout the entire effort, with design, of course, preceding programming. A reconstruction of the ARRCs team structure based on interviews, sets up this relationship:

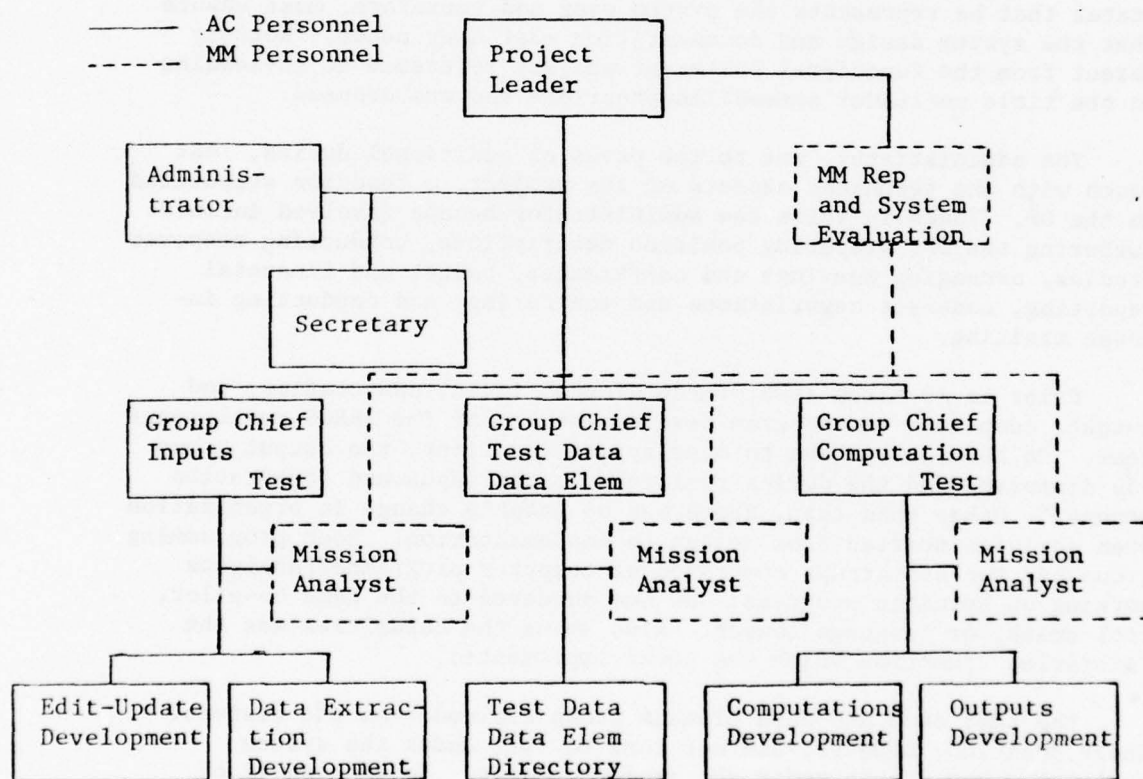


Figure 4. The Evolved Team

As in the previous diagrams, dashed lines are used to portray the MM positions and functions and the solid lines, those positions filled by and functions performed by AC programming personnel. The titles chosen were those that most nearly defined the positions as described by the individuals holding them.

The project leader did not function as a programmer in the conventional sense, so the term chief programmer used in the DP was not used. The associated functional statement was accurate, however, as he was in charge of the total project, was instrumental in the design, development and implementation of the system; and provided leadership, reviewed work and offered constructive criticism.

The MM project officer functioned to a large extent as liaison between the logisticians in the various cognizant offices of the Materiel Management areas at the SM-ALC and the development team. As senior MM representative, he did exercise some measure of control over the mission analyst personnel. Among other things, the DP states that he represents the system user and therefore, must ensure that the system design and documentation meet user needs. Notably absent from the functional statement was any reference to co-leading so the title co-leader seemed inappropriate and was dropped.

The administrator, due to the press of additional duties, lost touch with the technical aspects of the project, a function stipulated in the DP. Tasks in which the administrator became involved include: authoring the DP, preparing position descriptions, conducting manpower studies, arranging meetings and conferences, budget and financial reporting, contract negotiations and monitoring, and conducting in-house training.

Prior to 29 March 1976, three groups, input, computation, and output, comprised the program development arm of the ARRCs development team. On that date, "due to overlapping functions, the output group was dissolved and the duties realigned to the input and computation groups." Other than that, there was no notable change in organization when activities shifted from design to implementation. Each programming group had two sub-groups comprised of computer programmer/analysts working on specific programs. No one answered to the name co-pilot, tool smith, or language lawyer. Also among the casualties was the "librarian" function which was never implemented.

The test data and data element group evolved from the testers. Their position, however, did not come to rest under the system evaluator, but rather under the project leader. In section 12 of the DP (ARRCS Program Test/Certification Plan) the function of the test group is spelled out in detail including the requirement for validation and authentication of program/modules. In the final analysis the test group produced test data only and individual

programmers and group chiefs performed the test group function in conjunction with the system evaluator. The test group also provided the Data Element Directory and a computer program that edits, updates, and retrieves the "automated" DED.

Though Mission Analysts were nominally assigned to each group, in practice they became a mobile resource used by whatever team had a requirement for mission data or required research on specific mission related issues. As might be surmised, Mission Analysts were deeply involved in identifying and defining data elements.

Walk-Thru's.

The walk-thru was to be practiced for the life of the project. All work was to be reviewed and the principle function was described as error detection. Walk-thru's as employed in the ARRCs development effort could be placed in two broad categories, design and coding. Design walk-thru's were normally held at the behest of either the project leader or the group chiefs. Coding walk-thrus were conducted only within the group and then at the request of the group chief or individual programmer. No formal procedures were established and scheduling was accomplished on an "as desired" basis.

Since each walk-thru was in essence an AdHoc gathering, with no specific guidelines as to recording, assignment of action items, or subject definition or limitation, no complete record is available. However, from the memos that are available, one issue stands out. The design walk-thru's were employed to a very great extent in arriving at a system design, rather than in detecting errors or deviations from established specifications. There were no limits set as to time or subject matter with the consequence that sessions ranged from a brief one half hour to a far less brief one and one half days.

Though the lack of structure in the conduct of walk-thru's might be cited as a short coming, the failure to have detailed, design-to specifications made the walk-thru's an essential feature of the development effort. It also made each participant, to either a greater or lesser degree, a designer of the ARRCs system.

Design walk-thru's held at the request of the project leader were often documented to some extent in memos. In addition to the project leader, the group chief of the group involved, one or more programmers, and Mission Analysts (often including the MM representative) were usually in attendance. On occasion the chief of the other development group attended as did representatives of the test data group.

Design walk-thru's held within each group were usually limited to group members and were, on the average, shorter in duration and more limited in scope.

Coding walk-thru's were held within the group (if at all) and were basically a one-on-one situation, often, but not always, involving the group chief. Of the coding walk-thru's conducted, many were initiated after program bugs had been identified rather than as a program step-thru prior to compilation or execution.

Of fifteen team members responding to the question "Did the walk-thru's provide a significant service...?", ten answered in the affirmative, three stated that "some" rather than "significant" service was rendered and two allowed that only minor benefits were derived. Of the nine team members actively engaged in coding programs, three had not employed line-by-line walk-thru's at all, and only two had "faithfully" adhered to the "line-by-line coding review."

As can be seen from the above, walk-thru's were employed primarily in the design area and used as their focus of attention the HIPO diagrams showing program inputs, processing, and outputs. Not all HIPOS were "walked-thru" in team sessions, the activities of the computation group receiving greater attention. It was generally reported that group walk-thru's resulted in an openness within each group. However, team walk-thru's judging by the correspondence available, were not seen in the same light and were on occasion regarded as an intrusion or evaluation.

Though some problems were encountered in the procedures, or lack of procedures, followed for walk-thru's and though not all HIPOS and even less code was reviewed, the use of walk-thru's was considered, in aggregate, as the most significant factor in contributing to the reduction in program bugs.

HIPOS.

HIPO diagrams were produced for about 80% of the programs/program modules written. An administrative procedure in the form of a memorandum was developed to facilitate correction of HIPO charts typed on the Mag Card II typewriter. However, though there was a level of standardization within each group, as to scope and content, there was no standardization between the groups. There was an area of agreement, however, that, though HIPOS (as employed in the ARCS effort) provided utility in defining program function they were of little value in producing program code. This delineation of program function must have provided significant service however, since the use of HIPOS was ranked as the second most important factor in the reduction of program bugs.

Development Support Library.

Without the position of librarian being filled, the maintenance of a central development support library fell by the wayside. The librarian was to have maintained office procedures, a full set of documentation, program listings, references to the machine library, test results, etc.

Office procedures: These were generally disseminated in the form of memos. They were not classified or grouped as to subject matter and no numbering system or subject index was employed. Procedures were "bucked around" to be initialled and group chiefs usually retained copies of the memos in a general file. The single exception to this was the establishment of a three ring binder that was to contain data relative to operations on the IBM 370 computer at California Health Data Corporation. It was started with an initial seven items but failed to grow beyond the original size.

Documentation: Documentation consisted primarily of the FD, data element dictionary, DP, HIPOS and compile listings. Development group chiefs each had copies of the first three as well as compile listings and HIPOS for the majority of programs in their area. The FD and DP were in the possession of any team members having an interest or need for them. Up until the time of the Phase I certification other documents such as the data requirements document, system/subsystem specification, data base specification, etc. listed in the development plan and described in DODM 4121.17-M were not produced though some halting steps may have been taken in that direction.

Top Down Design.

This technique as described in the DP was followed. The system was analyzed by function, the system structure was developed, and the design was evaluated. Out of this effort came the "Functional Relationship Chart[s]", Appendix 4 to the ARRCs functional description.

Top Down Implementation.

It is at this point that the greatest divergence from the techniques of top down design and structured programming takes place. The DP itself adds to the confusion for it states:

"In this method, programs or series of programs need not be developed in a sequential manner. An example user output before developing the edit runs. This enables the user to have output for training prior to delivery of the total system."

This is contrary to any other text the author has read on the subject, but yet it was, in large measure, the precursor of the actual implementation methodology followed in the ARRCs development.

An IBM text on the subject defined the top down approach as "patterned after the natural approach to system design and requires that programming proceed from developing the control architecture (interface) statements and initial data definition downward to developing and integrating the functional units."

It goes on to say:

"The top down approach requires that the data base definition statements be coded and that actual data records be generated before executing any segment which references them."

The team structure itself with an input group and computation group working simultaneously at defining and coding system components makes the top down implementation, as defined elsewhere than the DP, an improbability. Development was carried on simultaneously and communications between the two groups was considered poor in conveying design data. An interface problem did develop between the data base and computational program and required the development of a special read module late in the development cycle. There is some conjecture that the read module was the most effective method for interface in any event.

Structured Programming.

Though each programmer involved attempted to employ structured programming techniques to the fullest the results varied from programmer to programmer. The initial program was for most the major obstacle. Afterwards coding came closer and closer to reaching the ideal. In addition to walk-thru's and HIPOS, the use of coding conventions such as indentation and the elimination of GO TO's to the extent possible were ranked high as reasons for fewer program bugs. Though impossible to quantify there was general agreement that significantly fewer bugs did appear in the structured code as opposed to conventional code and that problems, in the form of bugs or specification changes, once identified, were easier to resolve.

Major Handicaps.

In addition to the constraints on system development listed in Section I, there were several major handicaps or obstacles confronting the development team. Some of these were external to the team and in this category I would place the already discussed non-availability of an IBM 370/dd computer on site, and the seemingly arbitrary

derivation of the Phase I completion date. Others were internal and include training, lack of the paraphernalia associated with software development activities, and an adequate definition/specification for the system under development.

Training.

Training, or rather the lack of it, was evident in three broad areas: 1) Target equipment to include COBOL differences, Job Control Language and the operating system; 2) Top down design/implementation, structured programming, and employment of all the tools associated with TDD and SP; and, 3) Requirement computation systems in general - this being important since each team member, to a greater or lesser degree, was expected to serve as analyst and system designer.

By no means was everyone handicapped to an equal extent by each of the above listed training shortcomings, but the team as a whole would have benefited greatly if this training could have been provided prior to the development effort. The DP addressed the first two areas.

Target Equipment Training.

Training on the target equipment was to be provided by the computer vendor. However, with the exception of a one day overview of DOS/VS (Disk Operation System/Virtual Storage) which was generally conceded as providing little tangible benefit, and an orientation conducted by the California Health Data Corporation which primarily addressed the CHDC environment, no other formal IBM 370 training was forthcoming.

The requirement for system engineering support though recognized early in the development cycle, was not met until development was nearing completion and then only in a limited fashion.

A system engineer performs largely in a consultant's role and can be employed in any number of ways. He is considered the authority on programming and can evaluate code for efficiency or analyze programs experiencing particularly perplexing bugs. His major attribute however, is his knowledge of the operating system and system utilities. This includes a complete understanding of job control language, system generation, access methods, and file formatting, dump interpretation, and the employment of programming, development, and operating aids, such as procedure libraries, copy libraries, generation data groups, debug language, etc. An ideal situation would be to have this expertise available in-house and though this training is admittedly expensive, the dividends it would pay would, in all likelihood, offset the costs involved.

TDD and SP Training.

Although the DP stated that 22% of the people assigned to the project had had a 40 hour course in the Software Engineering Techniques (top down design and structured programming) to be used, it had been almost two years since that training had been conducted and, more over, it had not, to any significant degree, been applied to a true life development effort. An in-house course was taught starting in early March. This was, according to the DP, intended as an overview and course duration was 16 hours. In-depth lectures to be provided just prior to the application of techniques within the development cycle never materialized, due undoubtedly to the crush of development activity. Though the conclusion reached may well be considered speculative, there appeared to be an underlying assumption that if an individual were "familiar with the language peculiar to the techniques" that their application might follow naturally, rather like knowing that open heart surgery is surgery conducted with the heart open. The radical departure of TDD and SP from standard practice was probably not appreciated. At any rate, it was not adhered to.

Requirements Training.

Six programmers attended a short course on the D041 system. This is the applications software capability employed by the Air Force in Requirements Determination. Since development of program specifications was a joint effort exercised through the walk-thru's, an individual understanding of what the system was to accomplish and how it was to accomplish it was far more important than programming in an environment where detailed specifications are provided programmers who then fill out the "specs" with the required code.

When posed the question "On a scale of 1 to 10, [with 10 being top score] how well did the training prepare you for the work ahead?" the average rating extended by the programmers was just at three.

Of the four Mission Analysts interviewed, only one had attended the D041 training and responded with an eight to the above question. Two Mission Analysts had worked on Requirements Systems for prolonged periods as system users and the fourth was starting pretty much from scratch.

Development Structure

Though a large development effort (a major portion of the Advanced Logistics System (ALS)) had recently been concluded at SM-ALC, there was an apparent dearth of procedures and paraphernalia usually found in a software development shop. Shop standards and conventions were not in evidence nor were there in-house procedures for Incident Reports, Technical Memos, Reporting and Recording Historical Data, Managing Development Libraries, etc. There were no

development oriented operating instructions that could be added to or modified as the unique requirements of a unique effort using unique (for SM-ALC) procedures unfolded. Though the DP alluded to much of this, the implementing instructions were never prepared, and in consequence, the data not collected, maintained, or evaluated, and procedures not defined, refined, and adhered to.

There was no central activity from which training could be obtained or through which training could be arranged. There was also no in-house activity to which operating system software problems were directed for solution.

In the final analysis there were no effective tools to measure development status and progress (or lack there of) and consequently, whether by design or default, the ARRCs effort was, in large measure, carried out in isolation.

System Definition

To grow is good, to grow like Topsy, now that's a different story. Initial ARRCs development was to take one year and require an aggregate of ten people. From there, it expanded to one year and nine months and 18 people. At that time, (FD development) the total system was fairly well defined in a general or overview form. Before FD publication, the time frame expanded to three years and one month based on the criteria already reported, and study is now underway to expand ARRCs even further to provide the capability to compute war reserve stocks. Important now is the lack of definition of each specific phase. Phase I entailed the production of the Requirements Computation Worksheet. On the ARRCs Phased Development Chart this was referred to as "Basic DO41" and no further advanced notice was given as to what "Basic DO41" embraces. Rather like Humpty Dumpty's retort to Alice in "Through the Looking Glass" it could be stated, "It means just what I choose it to mean, neither more nor less." In order to plan and to measure progress, a completion date must be forecast, specific milestones must be established, specific tasks must be defined, and the system must be adequately described. And the secret is not to do it in that order.

The ARRCs effort from a systems definition standpoint appeared to this novice in requirements systems to be quite cryptic. Two or three sentences provided sufficient input to extend the development effort by many man years. The lack of specificity may, on the one hand, give a feeling of comfort and could indeed be adequate if everyone understands and everyone will always be there but this is highly unlikely. On the other hand, lack of specifics is only postponeable up to a point and it is at that point that the chickens will most assuredly come home to roost.

System definition, the reduction to its constituent parts, and further redefinition to embrace the individual tasks involved in its development is essential.

The Target Date.

Given time, the lack of training could have been overcome for the problem was recognized in the DP. Also addressed in the DP were many of the areas identified as shortcomings under the heading, Lack of Unique Development Structure. Assuming that recognizing the requirement would have led to its fulfillment, the culprit became the arbitrarily arrived at Phase I target date that caused a time compression problem. Not that Phase IV date appears any less arbitrary, just comfortably years in the future. Working backward from a sacred date is rather like deciding a year in advance (or maybe six months) to have a baby on Christmas, only it's a lot less fun. Since no one had calculated, with any degree of precision, the gestation period of a "Basic DO41", nor would it have mattered much if they had, what was delivered on the due date would be it. It should be recognized that political aspects must of necessity often outweigh the purely technical considerations of software development. However, equal recognition should extend to the realization that an apparent gain through early system delivery may well be offset by system or documentation shortcomings, or worse, become an aggregate loss when the system must be maintained or even re-developed at some not too distant date.

STATISTICAL RECAP

Statistics (1 Jan through *22 Jul 76)

Lines of Code Produced	15,988
Man Hours	
Programmer/Analysts/Admin Support	12,663
Mission Analysts	<u>4,275</u>
Total	16,938
Cost	
Salaries	\$190,461.57
Support	12,388.94
Computer	<u>14,145.35</u>
Total	\$216,995.86
Lines of Code Per Man Day	7.5
Total Cost Per Line of Code	13.57

* July figures are estimates.

SECTION III

RECOMMENDATIONS

General

In providing recommendations the author has an opportunity to appear omniscient, egged on by the gift of hindsight and the knowledge that he will not be responsible for implementing any of the suggestions made. Only one alternative is presented for each area perceived as requiring change. But many alternatives exist, one of which might entail maintaining the status quo, which could prove just as, or perhaps even more, effective than the recommendation provided. The author also realizes that certain choices are institutionally proscribed but indulges in the fantasy of "wouldn't it be nice." An example that comes immediately to mind is the establishment of due dates based on a realistic evaluation of the problem and the resources available to develop the solution. To omit addressing these matters would tend to discount their significance, however, political and structural considerations have a pronounced effect on most large development efforts. Too much or too little control, too few or too many resources may accrue when the strictly technical aspects of a strictly technical problem are subjected to non-technical "solutions."

I recognize that I have a built in bias that comes from attempting to apply what worked for me at other times, in other places to similar situations. Often several iterations occurred before I discovered a satisfactory solution and providing the reader this insight will not provide him or her the personalities that made a particular solution preferable to another. All recommendations I make, however, are not up-for-grabs. For example, I consider the establishment of standards and procedures and the utilization of formal task accounting/project monitoring techniques essential.

The major source of my recommendations, however, was the members of the APRCS development team who, during the course of the interviews, touched on every idea I put forth in the following paragraphs.

Organization

Applications Programming.

Though not addressed directly in the study, the creation of a single applications programming and computer program maintenance activity (programming shop) should provide major benefits. It would allow greater specialization in certain areas (librarian and technical writer for example) yet allow for greater flexibility in allocating resources. Standards and procedures would be easier to develop with

the elimination of two or three parochial views even though the immediate approach were something less than catholic. Enforcement of standards would be more uniform under one watchful eye than many.

Ideally, within the programming activity a flexible organizational structure would be attained to allow team orientation for major projects whereas program maintenance and minor development would be best served if area or machine oriented. Individual programmers would be moved between and among the various teams, not only depending upon workload but also to provide broadening experience and allow for the cross-feed of techniques and applications software capabilities data. This movement, essential to provide redundancy, must be carried out in a judicious manner.

Systems Programming.

The lack of adequate systems programming (often referred to as software specialist or systems engineer) support was evident in the ARCS development project. (pp 8, 27) I highly recommend the establishment of a strong systems programming activity (systems shop). From an organizational standpoint the systems shop could be placed either directly below the Director of Data Automation, a co-equal of the programming shop, or as one of its two components.

The systems shop performs a complex function from both a technical and administrative standpoint. Its major orientation should be that of service activity for the programming shop. In that light, it must be capable of performing the classic development support programming tasks outlined on page 27 of the text. In addition, the following duties should also be incorporated in its charter.

Training: A limited in-house capability to familiarize programmers with: changes to the operating systems (usually brought about by version changes); new hardware; solutions to recurring problems identified by system monitoring techniques, or computer operations; core dump interpretation; etc.

Provide the interface with ATC or commercial vendors in the acquisition and scheduling of AF provided or contract training.

Documentation: Serve as documentation authority to resolve problems in the documentation area including the interpretation of standards.

Technical Reference Library: Maintain the tech library. If extensive development is envisioned the Auerback or DataPro publications could prove a valuable addition.

Equipment Monitoring: Monitor hardware and systems performance to determine adequacy of equipment and software design.

Systems Housekeeping: Handle required housekeeping and utility chores such as disk dumps or reformatts, building procedure libraries, and passing judgement on assembly language programs to keep them out of sacred core.

Special Studies: Perform studies such as those leading to equipment substitutions/trade-offs, system configuration changes, and system conversions.

Training

Training, or the lack thereof, is documented on pages 27 and 28 of the text. Both the target equipment and the programming techniques to be employed were almost totally new to the programming team and the background in requirements systems was also, in large measure, sketchy. Training requirements must be defined and, to the extent possible, met prior to development. The use of ATC mobile teams should be considered and training conducted on a continuing basis to create and upgrade skills and develop a pool of resources. In-house training should be limited to that described under the heading of Systems Programming. From a cost standpoint, ATC provides the best buy but specialized training on specific hardware will, in all likelihood, require vendor support and a substantial expenditure of funds per student enrolled. To forego training may prove to be a poor economy when support must be contracted for, as in the case of ARRC Systems engineering requirements, or when time and money saving equipment, techniques, and procedures never enter the domain.

It should be noted that the employment of a flexible organization described on a preceeding page would provide a level of training on in-house software capabilities as well as facilitate the exchange of techniques, insights, and procedures between individuals.

Team Arrangements

Though the close association of MM and AC provided many tangible benefits (pp 18-23), there are some problems that must be guarded against. When the programmer/analysts get in bed with the system user the two things that usually result are:

1. The programmer/analysts, being more familiar with computer system development, will start doing the user's work of defining requirements.

2. The user, seeing the trials and tribulations of giving birth to a software capability, will start making the required excuses for late delivery or accept systems that fall short of requirements.

Mission analysts should, if possible, be kept as an entity with assigned areas of responsibility. Tasks must be formalized and formal procedures must be adhered to. This is not to establish two warring camps, for close, harmonious relations are very essential but a clear delineation of tasks and duties is no less so.

Documentation

Program and system documentation are the perennial problems of most development efforts with which I have been associated. ARRCs was certainly no exception (pp 16-18, 24, 25, 26). The best way to assure poor documentation is to attempt to overdocument. Confronted with a seemingly impossible task, most people will avoid coming to grips with it. It should be noted that more people avoid the mountain "because it's there," than climb it.

The first characteristic of good documentation is that it's useful. Obviously if it's not useful it's useless and should not have, or may well not have, been produced in the first place. If all it does is fill a square, fill the square with a no-op and move on.

Assuming that it has passed the first test the next test is, "is it current?" Out of date documentation can be worse than no documentation at all, so machinery must be established to assure its continued currency.

If DODM 4120.17-M is to be used, I would strongly recommend that the phrase

...ensure that the information that is necessary for that project has been included and that the information will be useful when the project is implemented. [The emphasis is mine.]

appearing in paragraph 1.3 of that manual serve as the guiding principle, not only in selecting which documents will be produced, but also the depth to which they will be developed.

In those cases where a computer product could suffice, as in the case of the data element directory, a standard capability should be developed with a product suitable for display on an 8 x 10 1/2 inch sheet of paper. Exception updating and change identification should also be incorporated.

In the case of the development plan, a comprehensive set of operating instructions could replace much of the DP allowing it to cover only the project unique aspects of the development effort.

One document that should be developed early is the documentation plan. (An OI should describe its form and content.) Each contributor must be identified by name or title and the exact form of his or her input should be specified. If, for example, a HIPO diagram and program abstract are required for each compiled module of the system, these should be presented by the programmer in final form when the program/module enters local certification testing. The required bits and pieces of documentation gathered as the development cycle progresses should form the nucleus of the documentation effort. If extensive development work is contemplated, the employment of a technical writer should be considered. The function of the technical writer is not to relieve team members of their individual writing responsibilities but rather to tie up the bits and pieces, judiciously gathered, into the final written output.

Software Engineering Techniques

The Development Team.

As documented in the text (pp 18-23), the team did not assume the characteristics described in the DP. In developing a system top down it would be advantageous if the various programming groups could be brought into play as the higher modules reached completion rather than carrying on in the more conventional pattern with input and output functions being developed side by side. (See pages 18-23.) The de-intimatizing of the AC/MM relationship has already been discussed. In ARRCs, the MM function should operate alongside the programming function and not be imbedded in it.

In a "chief programmer team" the chief programmer, by definition, programs. If some other technique or relationship is thought preferable, and it might well be, the titles and graphic structure should conform to the reality.

The loss of the librarian function with the concomitant loss of the development support library removed one of the major tenets of TDD/SP and one of the more interesting features of that concept (p. 25). A strong librarian could have divined a measure of progress and assured greater standardization between development groups.

I recommend that in all future TDD/SP development the librarian position be activated and given to a highly competent individual. The librarian should not be used as a "go-fer" but should, in addition to those tasks defined in the ARRCs DP, take charge of the DED, task accounting, and its logical consequence, project monitoring.

Walk-Thru's.

Along with several other procedures, walk-thru's should be governed by a formal procedural statement. Content, time allotted, attendance, recording, assignment/disposition of action items, etc., should all be addressed. (pp 23-24)

Standards/Procedures

Shop standard must be developed and enforced (pp 28-29). They should address, at a minimum:

1. Task assignment
2. Programming techniques
3. Coding conventions
4. Documentation

Procedures must be established to facilitate:

1. Requirement submission/evaluation
2. Work scheduling approval/disapproval
3. Design review
4. Project/task monitoring
5. Testing/certification

In developing standards, procedures, and operating instructions managers will more often than not assign this task to an individual who may have displayed an ability to write or simply to someone who is not busy with more pressing duties. Though the manager may believe he has something more important to do, he is generally wrong. These are HIS standards, procedures, and operating instructions and dictate how his team or office will function. He must think it out and relate it to his management or organizational philosophy and the standards he believes essential to enforce. If he doesn't compose every word, he must at least provide explicit guidance on what is to be addressed and how it is to be executed.

Two areas listed above require some amplification. These are: task assignment and project/task monitoring. Though verbal assignment of tasks may prove adequate for small projects, the more people involved and the more complex the problem, the more important it becomes that a formal mechanism be established to define individual tasks. Procedures must also exist that allow feedback concerning time estimates, both prior to, and during task accomplishment.

The use of a formal written procedure in assigning jobs is of great value. Since it requires the originator to define exactly what he wants it will normally result in a more thoroughly thought out request and, to a great extent, eliminate the misunderstandings

that result from verbal orders. Since even the largest development project must be divided into a collection of smaller, more manageable tasks, this should provide the requisite measure of bookkeeping to identify what is finished, what is in the works, and what remains on the drawing boards.

In assigning a task it is important to obtain inputs from the individual made responsible for its accomplishment. Estimates of the work involved (usually in man days) planned start date, and forecast completion date, are essential for planning purposes. This is especially true if a critical path has been identified and the job is astride it. Along with identifying the development status, (progress or lack thereof) compilation of this data will tell a manager where manpower resources are fully employed and where reserve manpower exists. Each individual's workload can also be determined. Periodic feedback must be part of this system for early completion of individual jobs would allow additional work assignments to be made whereas forecast slippages could point up the requirement to employ additional resources or reorder priorities.

I recommend the development of a task assignment procedure that ties in closely with documentation requirements. Some level of narrative description should be employed though HIPO's and pseudo code may serve in defining the more technical aspects of a problem. Task assignment, in addition to providing the nucleus of the required documentation should also describe the documentation to be developed as part of the assigned task. e.g. listing of edit error messages, program abstract, file format, etc.

I recommend the use of an automated task accounting system but have strong reservations about a time accounting system which generally reveals only that there are eight hours in each work day and that everyone reporting is fully employed.

Book Two, Section 19, of this study contains an extract from a task accounting system which has evolved over the past eight years at HQ PACAF. It has been found to be exceptionally useful in managing projects, accounting for manpower expenditures against specific systems, and providing visibility for both the Chiefs and the Indians.

Administrative Procedures.

In addition to a dearth of standards and procedures, office instructions were also lacking (pp 25, 28). Administrative file assignment instructions and a file plan should be developed. The use of the Dictaphone could be enhanced by the development of standard formats designed to assure that all required data is addressed in those areas where some measure of standardization can be achieved.

As an example, if historical data is to be maintained on specific projects, a format such as that appearing in figure five would aid the secretary in both preparing and filing the report and the originator in its development.

Historical Report

Format # _____ Historical Report # _____

System _____ Date of Report _____

Time Span Covered _____

Originator _____

Subject _____

Replaces/Aggregates Report(s) _____

Description of Event(s) _____

Significance _____

Attachments _____

Figure 5. Sample Format.

System Definition/Target Dates. (pp 29-30)

After all the prose has been put to bed and the viewgraphs shoved in the bottom drawer, after all the glowing and glaring generalities have passed on, almost every software system still ends up composed of the same basic building blocks; Input Formats, Elements of Data, Edits, Updates, Data Bases, Support Files, Data Manipulation Programs, Data Retrievals, Submission Modules, and Occasionally a Control Program.

A system must be defined to at least that degree before a realistic time estimate can be derived. An analyst must work with the user in defining the system to that level of abstraction for neither individual would normally embody all the expertise or have the identical vision of what to do and how to do it. A user who can dictate programming techniques does so at his own peril but a user who accepts without question the analysts view of what is best for him from a product standpoint, is equally imperiled.

Individual programs may be defined in a short narrative with the qualification, easy, average, or hard. A best guess of work (man days/man weeks) required can then be derived. The estimates for the entire system can be aggregated and, if you're cautious, multiplied by two, and, if you're realistic, multiplied by three.

What has been accomplished is an understanding of what, in general terms, will be produced. The analyst knows, and the prospective user knows. If the method of estimating time appears arbitrary it's only because it is. Yet there are benefits to be derived, for you have at least established times that can be attached to dates and tasks that can be assigned to people. The accuracy of your estimating technique can be checked empirically as the work progresses, as well as before the fact when the macro defined task receives a micro examination from the programmer assigned to bring it to fruition. The refinement of estimates is an on-going procedure that requires a task accounting system of some sort to be effective. The application of resources can then be varied to meet critical dates/handle routine chores/or get ahead of schedule.

In all likelihood, a pattern will emerge showing more time required for one function than originally estimated while other activities take less. The multiplier can be varied, up or down, to account for this and overall estimates refined until any deviation from forecasts becomes insignificant, or at least, easy to manage.

It must be stressed that system definition to the level described, macro time estimation, task identification and assignment, task accounting and feedback, and estimate refinement, are all part and parcel of a total system of project management. Each team builds up its own identity and momentum but once a team has functioned as a unit its productivity can be charted and predicted with reasonable accuracy. In this way the team has developed the due date and changes in direction that cause delays or project extension can be easily identified.

It's nice to know what you're supposed to be doing. This procedure assures that each individual does, and, analogous to the mirror on the bedroom ceiling, it allows a person to look up from time-to-time to see how things are going.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 14 SM-ALC/ACD-76-04	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER 9
4. TITLE (and Subtitle) 6 Software Engineering Techniques in Computer Systems Development,	5. TYPE OF REPORT, PERIOD COVERED Case Study - Final Report.	
7. AUTHOR(s) 10 John H. Lehman	6. PERFORMING ORG. REPORT NUMBER 12	
9. PERFORMING ORGANIZATION NAME AND ADDRESS Data Automation Branch Sacramento Air Logistics Center/ACD McClellan Air Force Base CA 95652	8. CONTRACT OR GRANT NUMBER(s)	
11. CONTROLLING OFFICE NAME AND ADDRESS Data Automation Branch Sacramento Air Logistics Center/ACD McClellan Air Force Base CA 95652	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS	
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) 12 51 p.	12. REPORT DATE 11 15 December 1976	
	13. NUMBER OF PAGES 40	
	15. SECURITY CLASS. (of this report) UNCLASSIFIED	
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE		
16. DISTRIBUTION STATEMENT (of this Report) Approved for Public Release - Unlimited Distribution		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) Approved for Public Release - Unlimited Distribution		
18. SUPPLEMENTARY NOTES Prepared in cooperation with members of the development team for the Allied Recoverable Requirements Computation System (ARRCS) at McClellan Air Force Base, CA.		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Software Engineering Techniques: The Development Team; Walk-Thrus; HIPOS; Development Support Library; Top Down Design; Top Down Implementation; Structured Programming.		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This case study centers on the effort involved in the development of a major logistics software system. (Continued)		

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

410189

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

20. Abstract (cont)

The development effort was unique in several major respects. It is divided into three general groupings. The first is historical and describes in the most general terms the system itself. The history predates the development effort by some months and terminates roughly with the certification of its initial phase.

The second part centers on a review and evaluation, not of the system, but of the procedures followed in its development. This evaluation is almost totally derived from commentaries, both written and oral, of the participants.

The final third of the text provides suggestions which the author hopes will prove useful in the development of the future phases of the System or totally new software development efforts undertaken at the Sacramento Air Logistics Center.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)