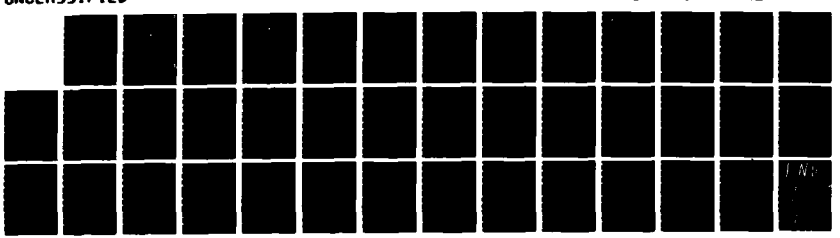
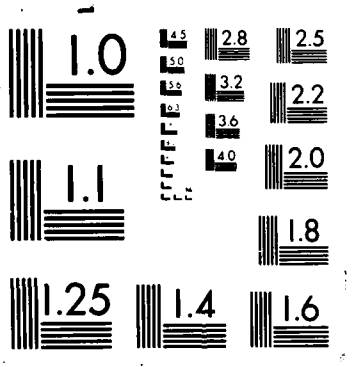
AD-A181 819 ALTERING THE APPLICATION OF THE TRADITIONAL SYSTEMS DEVELOPMENT LIFE CYCL (U) AIR COMMAND AND STAFF COLL MAXWELL AFB AL M L CHASE APR 87 ACSC-87-0485 1/1

UNCLASSIFIED F/G 12/5 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3

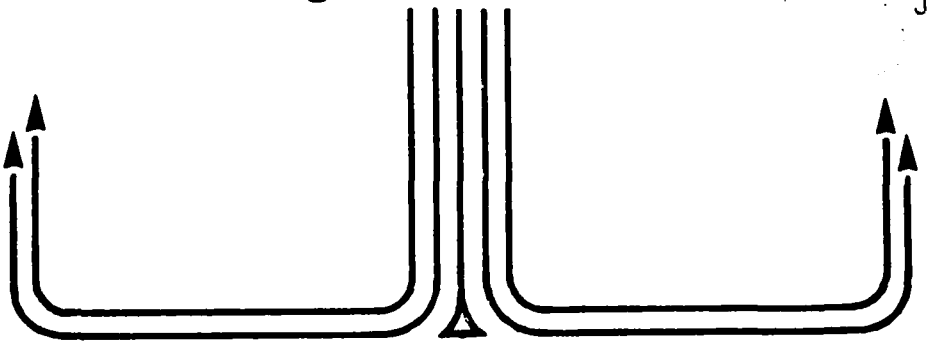
AIR COMMAND AND STAFF COLLEGE

STUDENT REPORT

ALTERING THE APPLICATION OF THE TRADITIONAL SYSTEMS DEVELOPMENT LIFE CYCLE FOR AIR FORCE SOFTWARE PROGRAMS

MAJOR MICHAEL L. CHASE 87-0485

"insights into tomorrow"



DISCLAIMER

The views and conclusions expressed in this document are those of the author. They are not intended and should not be thought to represent official ideas, attitudes, or policies of any agency of the United States Government. The author has not had special access to official information or ideas and has employed only open-source material available to any writer on this subject.

This document is the property of the United States Government. It is available for distribution to the general public. A loan copy of the document may be obtained from the Air University Interlibrary Loan Service (AUL/LDEX, Maxwell AFB, Alabama, 36112) or the Defense Technical Information Center. Request must include the author's name and complete title of the study.

This document may be reproduced for use in other research reports or educational pursuits contingent upon the following stipulations:

-- Reproduction rights do not extend to any copyrighted material that may be contained in the research report.

-- All reproduced copies must contain the following credit line: "Reprinted by permission of the Air Command and Staff College."

-- All reproduced copies must contain the name(s) of the report's author(s).

-- If format modification is necessary to better serve the user's needs, adjustments may be made to this report--this authorization does not extend to copyrighted information or material. The following statement must accompany the modified document: "Adapted from Air Command and Staff Research Report (number) entitled (title) by (author) ."

-- This notice must be included with any reproduced or adapted portions of this document.

REPORT NUMBER 87-0485

TITLE ALTERING THE APPLICATION OF THE TRADITIONAL SYSTEMS DEVELOPMENT LIFE CYCLE FOR AIR FORCE SOFTWARE PROGRAMS

AUTHOR(S) MAJOR MICHAEL L. CHASE, USAF

FACULTY ADVISOR MAJOR TERRY BROOKS, 3823 ACSC STUS

SPONSOR MR. JOHN D. HENNE, GM-13, HQ SAC/DOCR

Submitted to the faculty in partial fulfillment of requirements for graduation.

AIR COMMAND AND STAFF COLLEGE
AIR UNIVERSITY
MAXWELL AFB, AL 36112

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS: NONE
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): 87-0485	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: ACSC/EDCC; 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code): Maxwell AFB AL 36112-5542	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code)	10. SOURCE OF FUNDING NOS. — PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
11. TITLE (Include Security Classification): ALTERING THE APPLICATION OF THE TRADITIONAL	
12. PERSONAL AUTHOR(S): Chase, Michael L., Major, USAF	
13a. TYPE OF REPORT; 13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day): 1987 April; 15. PAGE COUNT: 36
16. SUPPLEMENTARY NOTATION: ITEM 11: SYSTEMS DEVELOPMENT LIFE CYCLE FOR AIR FORCE SOFTWARE PROGRAMS	
17. COSATI CODES — FIELD / GROUP / SUB. GR.	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

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

The military strategy of the United States depends heavily on maintaining a qualitative superiority over any and all potential adversaries. By maintaining this qualitative superiority the US is able to compensate for a wide variety of real or perceived shortcomings present in its military forces. This dependence has resulted in an increased emphasis on advanced technology and its associated software programming. The integrity and responsiveness of our software is being jeopardized by its high costs and an ever growing shortage of qualified software professionals. Although programs life the DoD's STARS program have recognized these problems, progress has been slow to reverse the trend. One area that deserves closer examination is the process the Air Force uses to conceive, develop, and implement its software. This process is known as the traditional systems development life cycle. This project will examine in detail the traditional life cycle, the factors affecting it, its economic foundations, and most importantly ways in which it might be altered.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☒ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL: ACSC/EDCC Maxwell AFB AL 36112-5542	22b. TELEPHONE NUMBER (Include Area Code): (205) 293-2483; 22c. OFFICE SYMBOL

DD FORM 1473, 83 APR — EDITION OF 1 JAN 73 IS OBSOLETE

UNCLASSIFIED

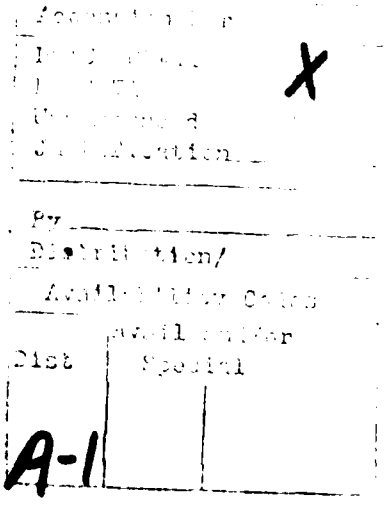
SECURITY CLASSIFICATION OF THIS PAGE

PREFACE

The military strategy of the United States, particularly in the arena of deterrence, depends heavily on maintaining a qualitative superiority over any and all potential adversaries. By maintaining this qualitative superiority the US is able to compensate for a wide variety of real or perceived shortcomings present in its military forces. This dependence has resulted in an increased emphasis on advanced technology, and its associated software programming. This has made software one of the most critical links in our military armor. However, the integrity and responsiveness of our software programming is being jeopardized by its high costs and an ever growing shortage of qualified software professionals.

Although programs like the DoD's Software Technology for Adaptable, Reliable Systems (STARS) have recognized the problems associated with software programming, progress has been slow to reverse the trend. Additional action must be taken to rectify the situation and increase the productivity of Air Force software programs. One area that deserves closer examination is the process used to conceive, develop, and implement software programs. This development process has been used since the earliest days of software programming and is generally known as the traditional systems development life cycle. While technology has been improving by leaps and bounds, the software development process has remained relatively static.

This project will examine in detail the traditional systems development life cycle, the factors affecting it, its economic foundations, and most importantly ways in which it might be altered. Altering the traditional systems development life cycle may be one of the only cost effective ways to improve the productivity of Air Force software programs. The consequences of failing to increase the productivity and reliability of our software programs is extremely grave and could have dire effects on Air Force readiness and US defense capabilities.



ABOUT THE AUTHOR

Major Michael L. Chase (B.S., United States Air Force Academy; M.B.A., University of North Dakota) has had extensive experience working with computer hardware and software. He has served as Chief, Management Information Systems for the Deputy Chief of Staff, Operations, Strategic Air Command, Offutt AFB, Nebraska. In this capacity he was the Senior Systems Analyst assigned to the Systems Integration Division, working the integration of all automation systems for SAC Operations. His primary area of expertise was working requirements definitions and long range planning. He also served as Special Assistant to the SAC Deputy Chief of Staff for Operations, advising him on all automation related initiatives. In this capacity he managed the acquisition, development, and maintenance of hardware and software for computer systems used throughtout the SAC Operations community. Major Chase is a distinguished graduate of the USAF Academy and winner of the Oliver LaGorce Award, given to the Outstanding Cadet in Geography. A distinguished graduate from Undergraduate Pilot Training, he was also awarded the ATC Leadership Award. He was a distinguished graduate of Squadron Officer School, and was a recipient of both the Wing Outstanding Achievement Award and Section Outstanding Contributor Award. He has also completed both Air Command and Staff College and National Security Management by correspondence.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

TABLES

FIGURES

EXECUTIVE SUMMARY

Part of our College mission is distribution of the students' problem solving products to DoD sponsors and other interested agencies to enhance insight into contemporary, defense related issues. While the College has accepted this product as meeting academic requirements for graduation, the views and opinions expressed or implied are solely those of the author and should not be construed as carrying official sanction.

"insights into tomorrow"

REPORT NUMBER 87-0485

AUTHOR(S) MAJOR MICHAEL L. CHASE, USAF

TITLE ALTERING THE APPLICATION OF THE TRADITIONAL SYSTEMS DEVELOPMENT LIFE CYCLE FOR AIR FORCE SOFTWARE PROGRAMS

I. Purpose: To establish the requirement for altering the traditional systems development life cycle and show that the use of a modular design will provide increased productivity and reduced software costs.

II. Problem: The military strategy of the United States to rely on qualitative superiority to maintain deterrence has placed an increased emphasis on high technology and its associated software programming. However, the integrity and responsiveness of the Air Force's software programming efforts are being jeopardized by its high costs and an ever growing shortage of qualified programmers, software project managers, and systems analysts.

III. Discussion: The importance of software to any military electronics system extends far beyond its functional role. The cost, reliability, and time required to field new systems are increasingly determined by software rather than hardware considerations. The cost of software to the DoD is currently $10 billion a year and is expected to increase to $30 billion by 1990. Software costs account for 5 percent of the Air Force budget, and are expected to climb to 10 percent by 1990. Additionally, software costs can exceed upward of 80 percent of the total system cost. Likewise, the demand for software professionals currently outstrips supply pipelines and has every indication of becoming a more intense problem. Since the demand for software and software professionals is not expected to

diminish, something must be done to improve the productivity and timeliness of Air Force software programs. One area that deserves closer examination is the process the Air Force uses to conceive, develop, and implement its software; a process that has remained relatively static since the early 1960s. Analysis of the phases of the life cycle process has shown that many factors impact software development. Among the primary factors are manpower, time control, early error detection, and a lack of overlap of the phases. While several different things may be done to improve the efficiency of the traditional systems development life cycle, the optimal solution may be the use of a modular design. Modular design is the dividing of a program into smaller segments, each of which evolves through the six phases of the life cycle. A module, however, is capable of functioning in a stand-alone mode. Each module builds onto previously implemented modules, and when all modules are implemented, they deliver the same capabilities and functions as the product developed using the traditional approach. Two modular designs are offered for consideration; a straight design where each module is completed before the following module is started or an overlap design where the next module is started before the previous one is completed. Either of the designs provides a degree of flexibility not attained under the traditional systems development life cycle, while delivering fairly significant time and cost savings.

IV. Conclusions: A modular design provides not only increased productivity and product quality, it can also produce cost savings of up to 10-15 percent. Additionally, it can also provide many intangible benefits; such as meeting the demand for increased time control, reduction of system changes, early detection of errors, reduced program size, reduction of maintenance costs, and increased programmer proficiency. All of which can improve the readiness of tomorrow's Air Force.

V. Recommendations: The Air Force must implement modular design wherever possible. This means evaluating programs currently being developed or in the process of being implemented, as well as future projects. Whenever possible manpower estimates and cost estimates should be based on modular design parameters. The Air Force must also develop and conduct training in modular design for all programmers, software project managers, and systems analysts.

Chapter One

INTRODUCTION

The military strategy of the United States to rely on qualitative superiority to maintain deterrence has placed an increased emphasis on high technology and its associated software programming. However, the responsiveness and integrity of the Air Force's software programming efforts are being jeopardized by their high costs and an ever growing shortage of qualified programmers, software project managers, and systems analysts. Although programs like the DoD's Software Technology for Adaptable, Reliable Systems (STARS) have recognized the problems associated with software programming, progress has been slow to reverse the trend (14:--).

The importance of software to any military electronics system extends far beyond its functional role. The cost, reliability, and time to field new systems is increasingly determined by software rather than hardware considerations. Software costs can exceed upward of 80 percent of the total system costs for some military electronics programs (6:13). The DoD is currently spending $10 billion a year on computer software and anticipates the demand to triple to $30 billion by 1990 (8:46). At $3 billion a year, spending for software accounts for five percent of the total Air Force budget, and is expected to climb to ten percent of the budget by 1990 (8:46).

Likewise, the demand for software professionals (programmers, project managers, and systems analysts) currently outstrips our supply pipelines and has every indication of becoming a more intense problem (8:46). Since it is highly unlikely that future systems will have reduced software requirements, other means of improving system performance must be developed. One area that deserves closer examination is the software development process. Better known as the software development life cycle, it is the set of activities that defines and describes how software is conceived, developed, and implemented. Since most software professionals utilize the same basic steps to define this development life cycle it is often referred to as the traditional systems development life cycle.

The purpose of this paper is to investigate this traditional systems development life cycle and determine whether altering the life cycle can provide economic benefits and increased productivity to the Air Force. This study will consist of an examination of the traditional system development life cycle; the

Air Force's application of that life cycle; the way the life cycle is depicted and how its costs are determined; some possible ways in which the life cycle might be altered; and finally the benefits of using an altered systems development life cycle. The consequences of failing to increase the productivity of Air Force software programs is extremely grave and can have dire effects on Air Force readiness and US defense capabilities.

THE TRADITIONAL SYSTEMS DEVELOPMENT LIFE CYCLE

With virtually all major Air Force electronic systems depending on computer software, it is absolutely essential that cost, quality, and productivity be optimized. According to Col. Kenneth Nidifer, Director of Mission-Critical Computer Programs for Air Force Systems Command, "Software has become a highly important force multiplier, even though this is difficult to express in terms of bombs on target. . . the very security of our nation depends on software" (8:46). While technology is lowering the cost of computer hardware, the sophistication of the tasks that software is called upon to perform is escalating the basic cost of software. Figure 1 illustrates how software costs, as a percentage of total systems costs, have been rising for DoD computer systems for the period 1955 to 1985 (18:13-14).

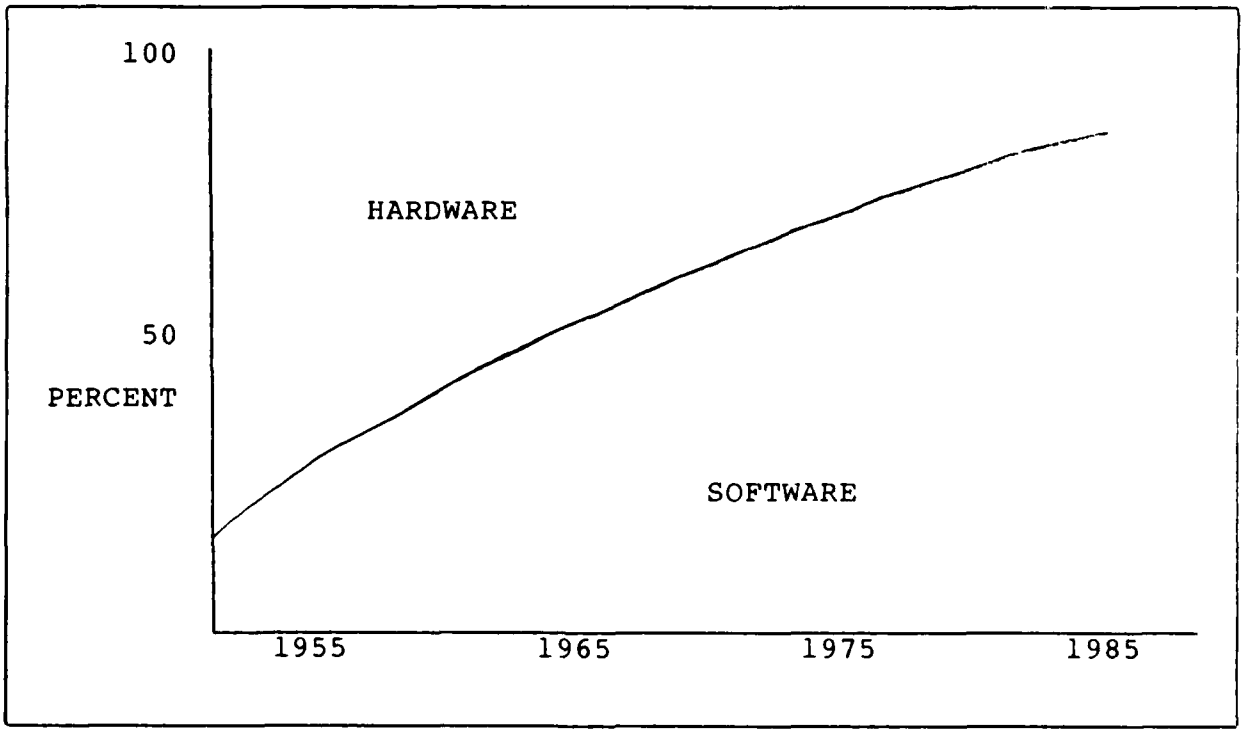


FIGURE 1. RELATIVE COSTS OF HARDWARE AND SOFTWARE

Definition of Phases

While software programming is critical to each computer system, the various phases of the software development process are not universally defined. Since the purpose of this paper is to determine how the systems development life cycle can be altered, a common ground must be established. What follows are this author's definitions of the various phases of the traditional systems development life cycle. The phases are a synthesis of the many phase descriptions available in computer literature, with major influence from the Planning Guide for Computer Program Development. The life cycle consists of six phases: 1) requirements definition, 2) specifications, 3) design and coding, 4) testing, 5) integration, and 6) implementation and maintenance (4:17-28). The description of each phase is listed below.

Requirements Definition

This phase includes the effort required to define and analyze the requirements for a software system, establish a software system description, and initiate the software development planning necessary to proceed with further development of the software system. This step may be repeated several times to ascertain that the true requirements are or have been stated and understood.

Specifications

This phase includes the effort required to convert the system requirements definition into a design; prepare a development plan that documents the cost, system performance specifications, and schedule estimates; as well as prepare a test plan. The preliminary design of the system is outlined. The user decides whether or not to proceed with the development of the software program at the conclusion of this phase.

Design and Coding

Following user acceptance of the system specifications, the design and coding of the software program begins. Actions could include logic flow diagrams, interfaces, data interchanges, data bases, files, records, and speeds; all of which are formulated and specified. Coding of the computer program is done during this phase, which results in translating the rather specific program requirements into appropriate sets of instructions.

Testing

This phase includes the effort required to support system testing and acceptance testing. The purpose of this phase is to ensure the system is in compliance with the stated system specifications.

Integration

This phase includes all efforts required to verify that the software system is in compliance with all project objectives, all deliverable items exist, and all reviews have been successfully completed. It usually includes real-time testing of all functions, to include stand alone operations which insure the system operates in and of itself. This phase also includes any efforts to incorporate the software with other packages or programs already in existence.

Implementation and Maintenance

In this phase system testing is completed, end users are trained, and the user organization is converted to the new software system. The software maintenance portion consists of correcting errors, making modifications, and enhancing system capabilities.

An important aspect of the software development process is evident from these descriptions--software is essentially a one time process. When a computer program has been developed (per these phase descriptions), the program and hence the software does not have to be "manufactured." The system is ready to function; with only modifications, enhancements, or corrections to be made. No additional development effort is required to make the software functional. As will be shown in Chapter Two, the Air Force uses a version of the traditional systems development life cycle to develop its software programs. The traditional life cycle has been incorporated into every facet of Air Force software development and has been utilized for over two decades. There are several additional problems; however, beyond the previously mentioned manpower and dollar shortages, that need to be considered and resolved. These problems are equally as significant in triggering a search to determine whether the systems development life cycle can or should be altered. These problems have surfaced in various research studies, as well as from the author's personal experience.

ADDITIONAL SOFTWARE DEVELOPMENT PROBLEMS

Automated Code Generation

Although automated code generation has increased the productivity of the coding process, and in many cases the maintenance function, reliance on automated code generation does not eliminate erroneous specifications. A research study by B. W. Boehm has shown that only 30 percent of post-delivery discrepancies occurred as a result of coding errors; the remaining 70 percent where caused by an erroneous design or specification (7:126). A study at GTE found that over 50 percent of all development hours were spent correcting errors resulting from faulty design (9:240). Automated code generation is fine

when utilized correctly, but reliance on it often leads to other errors that must be overcome.

Software Packages

Software packages often require major modifications, long-term commitments, hardware restrictions, and a forfeiture of flexibility. A recent GAO report to Congress showed an unusually large number of software packages are never implemented, and that those which do get implemented, are often delivered late, with excessive cost overruns (15:1-84).

Software Programmers

Programmers believe the reports which state good programmers produce 20-25 lines of debugged code per day (6:304-305). Long-term estimates easily become self-fulfilling prophecies. Slipped schedules have become the fashionable approach to increasing project manpower.

Manpower Impacts

While beefing up project manpower is used to bring a floundering project back on schedule, project leaders grossly underestimate the cost of adding programmers during the latter stages of program development.

Computer Jocks

Computer "jocks" design systems for computer "jocks"; users are not "jocks"! The operation of software programs needs to be understood by the end user, which is often impossible because of the way the program is designed and developed.

Development Philosophy

The philosophy that "there is not enough time to do it right today but plenty of time to fix it tomorrow," has caused considerable rework as well as a loss of user confidence in early release stages. Time constraints and productivity goals often force quality to be a secondary concern.

Systems Analysts

The systems analyst has a difficult time differentiating between user needs, wants, and dreams in trying to determine system requirements. The lack of formal "computer" education or relevant experience in the area of system's analysis has often led to a misdirected effort.

SUMMARY

The Air Force is dramatically increasing the amount of money it is willing to spend on software development, yet something must be done to meet the demands for increased productivity. While the sophistication of tasks software performs has increased dramatically, the Air Force is still using an antiquated approach to software development. This approach was used when our most powerful computers had performance capabilities that are less than today's desktop computers. Just as those computers have changed to meet the ever increasing demands of a "high tech" Air Force, so must the supporting software development process. This paper will examine that development process and show that altering it will provide both economic and productivity benefits to Air Force.

Chapter Two

THE AIR FORCE'S APPLICATION OF THE TRADITIONAL SYSTEMS DEVELOPMENT LIFE CYCLE

The previous chapter described the traditional systems development life cycle and how it has governed the development, implementation, and maintenance of software systems. This chapter will describe the Air Force's use of this systems development life cycle and the program management concepts it has spawned.

Air Force Regulation (AFR) 700-1 states the Air Force will "manage information as a critical resource" and "information systems will be implemented and managed to enhance Air Force war fighting capability" (12:2). To attain these lofty objectives, the Air Force has defined every facet of information system development, acquisition, and implementation. AFR 700-1 defines an information system as a combination of people, equipment, facilities, procedures, software, and other resources, organized to process information (12:2). It does not matter at what organizational level the information system is being utilized, it "will be managed according to the principles, policies, and procedures of life cycle management" (12:5). This implies the traditional systems development life cycle will be used to obtain and maintain effective information systems, at minimum cost, for the total life of the system. The Air Force's use of life cycle management principles begins with the conceptual needs (requirements definition) phase and ends with the disposal or replacement of the information system.

The Air Force references the systems development life cycle to provide a framework for the process by which the initial concept of a system evolves into a fully operational system. While the Air Force institutes various technical controls, through reviews and audits, the basic life cycle process remains fairly consistent with the traditional systems development life cycle described in the first chapter. AFR 700-4 Vol I, defines the phases of the life cycle as concept development, definition, program development, test, and operation (13:8). An analysis of Figure 1-1, AFR 700-4 Vol 1, shows the purpose of each phase is very closely aligned with the purposes of the six phases of the traditional life cycle previously defined (13:7-8). Concept development corresponds to requirements definition; definition with specifications; program development with design and coding; testing with testing; and operation with integration, implementation, and maintenance. Figure 3-3, Software

Development, AFR 700-4 Vol I, further breaks down this life cycle process and reveals, through an analysis of its baseline functions, how closely the Air Force systems development life cycle matches the previously depicted traditional systems development life cycle (13:19-20).

The purpose for establishing this relationship is to provide a common ground when discussing characteristics of the traditional systems development life cycle and the possible ways in which that life cycle might be altered. Again, it is important to remember there is no universally accepted definition of the systems development life cycle, and that the major differences discussed here are ones of semantics. It is also interesting to note that the application of the AFR 700-series regulations is applicable to all systems, whether they are stand alone or imbedded in larger systems.

During each phase of the systems life cycle, the Air Force has a single point of management with sufficient authority, responsibility, and accountability to direct the effective management of the program. Changes in management normally occur at prescribed points in the life cycle and since each phase runs subsequent to the next, have little overlap. The management plan will be transferred whenever there is a management change and will be updated with every change to the life cycle plan (12:5). Under the described Air Force systems development life cycle, as well as the traditional systems development life cycle, the process from start to finish is one long continuous development program, with delivery of the software system occurring only in the final step.

SUMMARY

The main point is the Air Force does use a version of the traditional systems development life cycle to develop its software programs and has been doing so for several years. Consequently, almost all of the problems associated with the traditional systems development life cycle, as uncovered in both civilian and military studies, would be applicable to either version of the systems development life cycle. As a result, and for the purposes of this paper, the two life cycles are considered the same and interchangeable. The next step is to accurately depict the traditional systems development life cycle, including its cost curves and cost estimates, and establish an economic foundation for altering the traditional systems development life cycle. This is the purpose of the next chapter.

Chapter Three

COST CURVES, COST ESTIMATES, AND THE LIFE CYCLE DEPICTED

This chapter will develop the economic foundation for altering the systems development life cycle, establish a baseline for software development costs, and identify the various factors affecting software development. The purpose of this chapter is not to provide an analysis for determining cost estimates but rather to view cost curves, cost estimates and the life cycle as a means of highlighting areas for improvement.

THE COST CURVE

As in many other projects, software program managers often ask: How long will it take? and How much it will cost? Software development is a dynamic process, and all efforts to depict the various cost curves will never be totally accurate; however, they can provide an insight into those areas which might need change.

The first step is to understand the pattern of the cost curve associated with the software development life cycle. Figure 2 depicts the generally accepted shape of the cost curve, with time shown on the horizontal axis and dollars of cost on the vertical (1:78). Through the first five phases the cost curve starts out initially high (peaking during design and coding) and then becomes downward sloping. This downward slope continues through the implementation phase but begins to climb as maintenance is required and the system becomes outdated.

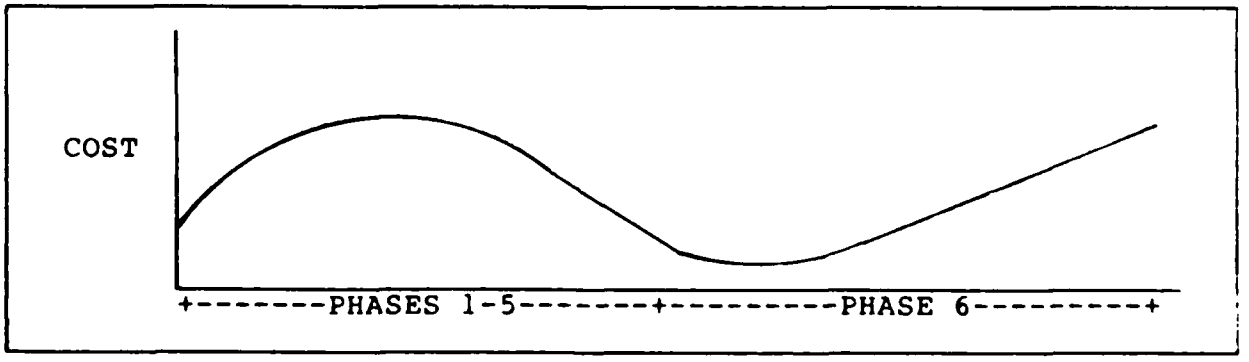


FIGURE 2. THE COST CURVE OF SOFTWARE DEVELOPMENT

Some general observations about this cost curve are: 1) as the software system life increases, development costs (phases 1-5) as a percentage of total system cost decrease, and 2) as total costs increase, the average cost per year that the system is in operation decreases--up to a point. Additionally, research by Peter Norden of IBM has shown that each of the six phases has its own well-defined cost curve (based primarily on manpower costs). The magnitude and duration of these individual curves are of a generally stable and predictable structure and can be exploited for project planning and control (20:1). The pattern of each smaller curve is in essence a smaller replica of the larger overall development cost curve. The curves climb, peak, and have support tails of varying length.

There is virtually no overlapping of any two phases. Consequently, each phase must be completed before commencing with the next phase. As a result, any delays which might occur in one phase very often impact all subsequent phases. This domino effect exists in many software development efforts and is often a major cause of program slippages. A graphic way of showing this aspect of the traditional systems development life cycle is through the use of a Gantt chart, as shown in Figure 3. The phases are shown on the vertical axis with time on the horizontal axis.

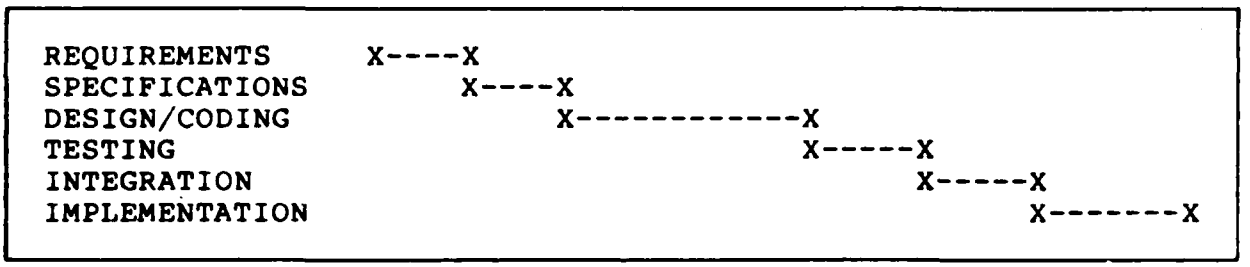


FIGURE 3. GANTT CHART DEPICTING THE TRADITIONAL LIFE CYCLE

ERROR DETECTION

One major cause of the above mentioned delays is the correcting of errors, better known in computer circles as "debugging." James Martin, in his book An Information Systems Manifesto, graphically shows how costs to correct errors sharply escalate as they pass from the requirements definition phase through to the implementation and maintenance phase (5:48). Figure 4 depicts this sharply rising cost curve. Martin states it could more than 1000 times more cost-effective to detect and correct errors in the requirements definition phase than in the implementation/maintenance phase. Further, the author's personal experience has shown that errors are often detected out-of-phase. Errors which occur in phase 1 (requirements definition) are often not detected until phase 6 (implementation

and maintenance), likewise errors in phase 2 (specifications) are not detected until phase 5 (integration), and phase 3 (design and coding) until phase 4 (testing). More specifically, design and coding errors are often caught in the test phase as the program simply will not function, where as errors in requirements definition may not be detected until the system is implemented.

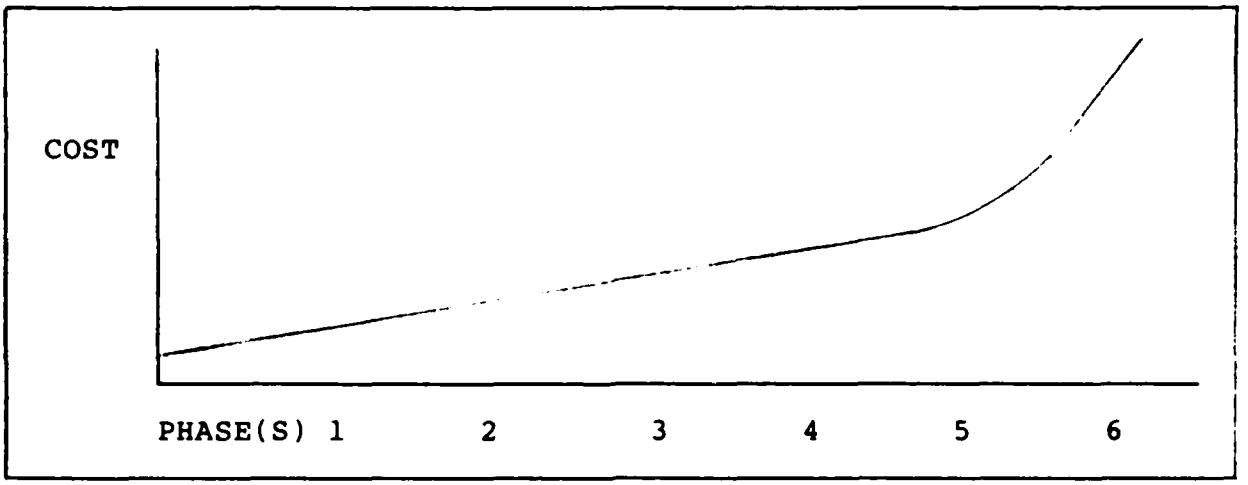


FIGURE 4. THE HIGH COST OF LATE FIXES

TIME SENSITIVITY

Each of the cost curves and figures discussed thus far has been directly related to time and that is not by accident. Time plays a critical role in the software development process. Schedule stability is often described as the most critical problem in software development. Software development is very time-sensitive, and development time specification is not the prerogative of management (as most of management believes), but rather belongs to the system (6:8). Management can iterate constraints and define a feasible schedule, but it can not always dictate the time for completion of a project. To get some idea of the time-sensitivity of software development, consider Table 1, generated by Lawrence Putnam in his study of software cost estimating. Using a simulation model, he was able to show time-sensitivity as a function of system size. Additionally, he stated this analysis could be used to describe any typical government software project (6:8-9).

LINES	TIME(MONTHS)	STD DEV	MAN-MONTHS EFFORT	STD DEV
15,000	12.9	0.6	34.7	6
50,000	21.6	1.1	376.5	60
100,000	29.1	1.4	992.9	152
250,000	43.1	2.1	3188.2	498
500,000	57.9	2.8	7782.3	1204

TABLE 1. TIME-SENSITIVITY AS A FUNCTION OF SYSTEM SIZE

What the table reflects is a very narrow time window. For example, a 15,000-line program has a standard error of 0.6 month. If management had set the time for completion at one year, it would require 0.9 month of excess time to be completed. In this case the probability of successful completion is only 7 percent (6:8). For a one year project some managers and users would accept a one month delay, but the case of the larger project is different. With a system having 500,000 lines of code, it would be very hard for management to guess 57.9 months for completion and 7782 man-months of effort. Because of cost and time considerations, many project managers would be more inclined to pick 48 months rather than 60 months, knowing there is a better chance of getting funded. Yet, 48 months is virtually impossible, as the probability of success is less than 1 percent (6:9). What this means is, if the development time is arbitrarily specified by managerial fiat, there is often a high chance the project will not meet its time or cost schedules.

Basic statistics further support this conclusion by saying in most cases the farther into the future a forecast or prediction is made, the less reliable the predication will be (3:89). In the case of software development, the author believes these predictions would more likely be inaccurate on the side of taking even more time to complete the project. The GAO report supports this position by showing a large number of government software programs are plagued with late deliveries and cost overruns (15:1-84). The only acceptable solution to this management dilemma is to set realistic time estimates and bias them for risk; or decrease the time of the project. The shorter the time element, the less the risk and the more reliable the estimate. Hopefully this study will show that altering the traditional life cycle can provide that opportunity.

OVERLOOKED FACTORS IN TIME (COST) ESTIMATES

While many studies have been made on how to improve the accuracy of the above mentioned time estimates, there are several factors which are very often overlooked or not considered when making time estimates. They are mentioned here; however, because

they have a definite impact on the reasons why the development life cycle should be altered and better yet, how it might be altered. The factors mentioned are primarily drawn from the author's personal experience and reflect areas which are necessary for more accurate estimates. The common formula for project estimates can be boiled down to: Contribution Level x Productivity x Number of Team Members = Effective Project Manning (11:9). Contribution level is the percent of a programmer's time to be spent on the project. Productivity level is the percent of a programmer's time that can be expected to be spent at work and not on leave or other non-project related activities.

The author believes, however, the common formula needs to address at least four other factors which are necessary for reliable time estimates. First is proficiency level, which characterizes the ability of assigned personnel to perform a given task. If appropriately expressed as a percentage, it could fit in well with the formula expressed above. Second is team turnover rate, which indicates the training and subsequent contribution of team member replacements. The turnover rate must be a function of time remaining until project completion. New team replacements immediately prior to implementation would have significantly more negative impact on the project progress than early turnover if all other variables are held constant. Third is a factor reflecting team addition rate, which represents an adjustment for personnel added to a team after specifications have been gathered. This adjustment may explain why additional manpower for floundering projects does not always produce the anticipated expeditious results. Finally a cohesiveness factor, which quantitatively describes the ability of team members to work together. In turn, cohesiveness facilitates optimal communication and assistance. Not only are the above four factors difficult to anticipate, little if any research is available for determining the direct impact they have. While this study will not address integrating these factors in with the standard estimating formula, it might show how altering the traditional life cycle can satisfy these concerns.

USER SATISFACTION

The purpose of any software development project is to develop software that meets the end user's requirements; however, the real question is: What is user satisfaction? My research has shown that meeting original requirements does not, in many cases, equal user satisfaction; rather original requirements plus changes more closely equals user satisfaction. In software development, changes are usually a function of development time and/or system dynamiticity. While it is difficult if not impossible to control system dynamiticity, development time can perhaps be controlled. If the time from requirements definition to implementation could be reduced, changes might be minimized. Minimizing changes can result in a product which would more nearly meet original user requirements. Additionally, rework

could be significantly reduced, and a product could be delivered that directs future changes rather than incorporating them.

MANPOWER TRADEOFFS

A majority of the costs associated with software development are manpower costs. The pattern of the manpower curve in a traditional software development life cycle is basically the same as the previously mentioned cost curve (manpower costs are the primary costs in most software development programs). This pattern is best known as a Rayleigh manpower curve. The parameters describing the Rayleigh equation can be arranged to provide an effort-time (manpower) trade-off law (6:43). This trade-off law states that small changes in development time result in very large changes in effort. As the number of people who must interact and work together on a project rises arithmetically, the number of interactions goes up geometrically. This results in more and more time being spent on human communication and less and less on productive work (6:35-43). The two ways to handle this problem are to limit the number of people working on a project at any one time, which will stretch out the time schedule; or to reduce the size of the project. If altering the life cycle could reduce development time, significant manpower costs and development costs savings might be attained. Again, time plays an all important factor.

SOFTWARE MAINTENANCE

Another area that plays an important role is the maintenance portion of the life cycle. Maintenance costs, when depicted as a percentage of the total system cost, increase with each year the system is in operation. Maintenance costs can be expected to consume up to 50-75 percent of the total system cost (19:vii). Edmund Daly found maintenance costs represented about 12 percent of the system cost in the first year and 46 percent after four years (9:236). Because maintenance costs do consume such a large percentage of total cost, a brief examination of these costs is appropriate. Additionally, maintenance enhancements are also subject to development approaches.

Software cannot be maintained because it is not subject to dilapidation or deterioration. Software does not require preventive or remedial maintenance and it does not have hardware maintenance characteristics. E.B. Swanson expressed it best when he identified three basic categories of maintenance (21:492).

Corrective--repairing an undetected error or specification.

Adaptive--modification necessitated by a changing environment.

Perfective--improving performance capabilities.

It should be noted that corrective action is also part of the test phase, and corrective action is necessitated by human error. Adaptive and perfective changes are part of the development process. Adaptive and perfective changes are a function of the dynamiticity of the software program, which with longer periods of development time increases the size of the task to be completed. The implication is clear; the satisfaction of a maintenance request could be enhanced by time limitations on the development process. To understand the impact of maintenance changes, consider Daly's findings at GTE. Some programs had as much as 70 percent of the original code changed in the first two years of maintenance, and there were instances where the average life of a line of code was only 18 months (9:240). Again, altering the traditional life cycle might provide opportunities to address these issues and resolve some of these problems.

SUMMARY

The purpose of this chapter was to define and analyze the software development process and its associated cost curves, and to build an economic foundation for altering the traditional systems development life cycle. Notation was made of the various factors affecting software development and their impact. Primary among these factors were manpower, time-sensitivity, early error detection, software maintenance, user satisfaction and lack of overlap of the phases. The next chapter will analyze this information and determine how it might be used to alter the systems development life cycle.

Chapter Four

ALTERING THE SYSTEMS DEVELOPMENT LIFE CYCLE

This chapter will draw some general conclusions from the information presented in the previous chapter and then use those conclusions as stepping stones to examine ways in which the systems development life cycle might be altered.

RESULTS OF ANALYSIS

The Pattern of Cost Curves

The total cost curve of the traditional systems development life cycle follows a consistent and set pattern. It starts high, peaks during the design and coding phase, becomes downward sloping through implementation, and has a steady climb in the maintenance phase. The cost curve of each individual phase also follows a similar pattern, although the first five phases (requirements definition through integration) do not demonstrate the dramatic support tail of the maintenance phase.

No Phase Overlap

Under the traditional systems development life cycle each phase is done sequentially, in a more or less stand-alone mode. There is very little, if any, overlap. Delays in one phase often have an adverse affect on subsequent phases.

Impact of Change

Software development is a dynamic process, with many factors subject to change. In many cases change has an adverse impact on the development process; usually requiring more manpower, time, and money be allocated for project completion. Changes are often a function of longer development time and overall system dynamiticity.

High Cost of Late Changes

Delays are primarily a result of errors or changes in the system design and coding. The costs associated with making changes can be rather dramatic, especially in the latter phases of the life cycle process.

Development Time

Simulation models show that errors in estimating the development time (which determine project cost) are subject to a greater error when the project is larger and takes more time. Manpower, resources, and time are allocated based on estimates made of the development life cycle.

Software Maintenance

Software is not maintained like most products. It does not dilapidate or deteriorate, nor does it require preventive or remedial maintenance.

Maintenance Costs

Maintenance costs comprise an ever increasing portion of the total system cost. The longer a system exists the greater the maintenance costs will be, both on a per year basis and as a percentage of total costs. Reducing maintenance costs can greatly reduce total system costs.

Time Sensitivity

Time control is imperative in all phases of software development. Reducing development time will provide greater stability as well as a wide variety of cost benefits. It can reduce change and turmoil, decrease manpower requirements, and improvement cost estimates.

Error Detection

Catching errors or making changes early in the life cycle process is cheaper than doing it in the latter phases. Additionally, error detection often occurs out-of-phase, with errors that occur in early stages not being detected until the latter stages of development.

Proficiency

Higher proficiency, lower programming team turnover, and lower user turnover will reduce development time and errors, while decreasing total system costs.

Program Size

Reducing program size (lines of code) reduces the amount of manpower required to complete the project, the time required, and total program costs. Reducing the size of the program also increases the accuracy of both the time estimates and total project cost estimates.

ALTERING THE TRADITIONAL SYSTEM DEVELOPMENT LIFE CYCLE

The above conclusions, as well as studies by Joseph Fox, Mitre, and TRW, have shown that something must be done to improve the productivity and timeliness of the software development process (2:--; 16:--; 17:--). Each offered some possible solutions to these problems and ways in which system efficiency might be improved. One area is the use of systems development tools. Another is the use of automated code generation, which has speeded up the coding process and in many cases the maintenance function. In the author's opinion, however, the optimal solution is the use of a modular design. Modular design consists of dividing a software system into smaller blocks of processes or modules, each of which evolves through the six phases of the traditional life cycle. While one module, capable of functioning in a stand-alone mode, would have fewer functions than the entire product of the traditional approach, it would provide the same capabilities and functions as the original system when all modules are developed and functioning together.

THE MODULAR APPROACH

To more completely understand and visualize what modular design is refer to Figure 5.

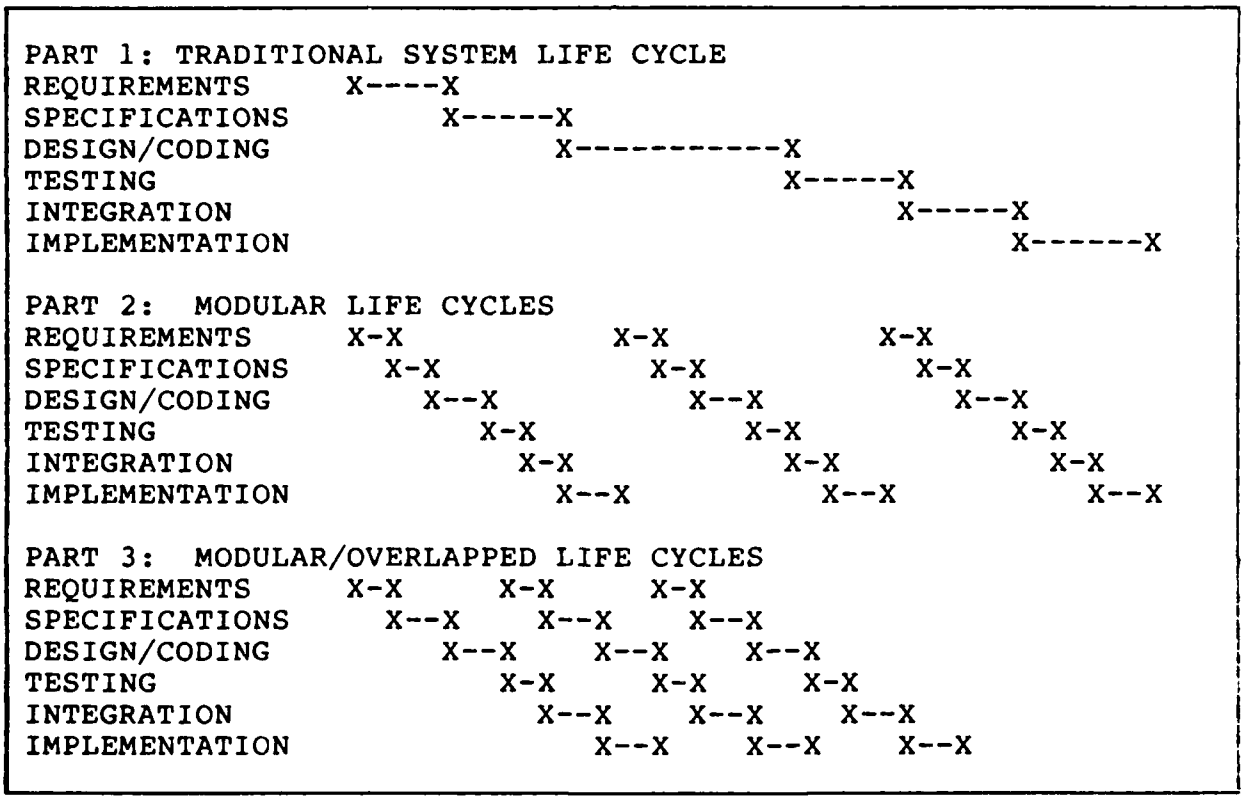


FIGURE 5. LIFE CYCLES DEPICTED

Part 1 is a Gantt chart depiction of the traditional systems development life cycle (it is a reproduction of Figure 3). Under this approach, as was described earlier, each step is done in sequence, with little or no overlap. Again, the possibility exists that delays in any one phase could result in delays in all subsequent phases. Parts 2 and 3 are Gantt chart illustrations of two possible modular life cycles. For purposes of this discussion, each project is divided into three modules. In real life situations there could be a multitude of modules, but the results would be basically the same. An assessment of modular design by Donald Latham, the Assistant Secretary of Defense for Command Control Communications and Intelligence, shows it is possible to break projects up into smaller modules and complete them under the basic guidelines set above (10:185).

In modular approach number one (Figure 5, Part 2: Modular Life Cycles), the project is divided into modules, with each module going through its own life cycle. This part is the same as under the traditional approach. Each phase is completed before the next is started, and the next module is not started until the previous one is completed. The time to completion of the last module (and consequently the entire project) is approximately the same as under the traditional approach. The first two modules, however, are completed and implemented well before the final product of the traditional life cycle approach is implemented. Therefore, this approach offers the end user some capabilities earlier than he would have had them under the traditional life cycle approach.

Under the second modular approach (Figure 5, Part 3: Modular/Overlapped Life Cycles), the project is again divided into modules, with each module going through its own life cycle. Under this approach, however, the next module is started prior to the completion of the previous module. In this case, the modules overlap such that the requirements definition of the next module is started when the design/coding phase for the previous module is completed. With this approach the entire project (all three modules) is completed in less time than is required for the entire project under the traditional life cycle approach. The modularity of this approach permits the overlap, whereas the structure of the traditional life cycle does not.

The techniques of modular programming are not new (where various parts of a program are developed in modules, but not meant to operate in a stand alone mode); however, the concept of modular design is. The very fact that it is new does create a few problems within the Air Force. Higher levels of management are not always known for their innovativeness and willingness to try new approaches. A large number of Air Force programs, however, appear to be perfectly suited to this modular approach. Since most Air Force software programs have a multiplicity of functions and often work at various levels, they could be easily broken down into modules. By prioritizing the various functions within the system being developed, programmers and project

managers could very easily institute modular design. Additionally, modular design could make reprogramming a much easier task as changing one module would have very little impact on other modules.

SUMMARY

To achieve the high rates of productivity, with the wisest allocation of dollars and manpower, it is imperative the traditional systems development life cycle be altered. The demand for increased time control, reduction of system changes, early detection of errors, reduced program size, reduction of maintenance costs, and higher proficiency rates all show the Air Force needs to reevaluate the process it uses to develop software. As was discussed, one such way to accomplish this is to use a modular design. Either of the two modular approaches presented solves many of the previously discussed problems and shortfalls. Additionally, they provide a degree of flexibility not possibly attained under the traditional systems development life cycle. The next chapter will highlight the benefits of altering the traditional life cycle and for using a modular design.

Chapter Five

BENEFITS OF USING AN ALTERED SYSTEMS DEVELOPMENT LIFE CYCLE

From the Gantt charts and discussion provided in the previous chapter, it is readily apparent there are several distinct benefits which result from using a modular design. The reductions in both time and manpower are especially significant. Additionally, there are several other benefits (both tangible and intangible) that can be accrued through the use of either of these two modular designs. The purpose of this chapter is to evaluate these benefits and to assess the impact they might have on the traditional software development process.

IMPACTS OF USING A MODULAR DESIGN

Reduced Manpower Costs

Manpower trade-off laws show that time and effort (manpower) are coupled and that small changes in development time can result in very large changes in effort (6:16). In the case of using a modular design, reductions in time will greatly reduce manpower and create a substantial cost savings. Time reductions, in and of themselves, also generate cost savings.

Impact. More personnel will be available for other projects, therefore, increasing productivity as well as organizational flexibility. In many cases the manpower factor is of greater concern than the dollars saved, as there may not always be additional manpower available. The associated cost savings will mean more dollars will be available to spend in other areas, or perhaps more significantly, a program may be funded when it might otherwise fail to make the funding cut.

Reduced Development Costs

Modularity leads to a commonality of design, which in many cases leads to a reduction in the total lines of code. John Snyders found that using a modular design (in two smaller programs) reduced the total lines of code by 15-25 percent (11:46). Reducing the number of lines of code can have a dramatic effect on reducing costs. Software cost equations show that reducing the lines of code by just 10 percent yields costs reductions of up to 27 percent from the original system price (6:7).

Impact. More dollars will be available to spend in other critical areas. Additionally, the cost of future maintenance might also be reduced. Maintenance could be limited to the module in question, thereby eliminating some of the ripple effect which naturally occurs as one change generates additional changes. Reducing development costs also means that the economic point of redevelopment occurs sooner. More frequent redevelopment is consistent with, and often triggered by, other industry advances in hardware and software.

Reduced Chance Of User Turnover

By reducing the elapsed time between requirements definition and implementation, it is implicit that the amount of user turnover will be reduced. It is much more likely that individual users who participated in the requirements definition and specifications phases will be present during delivery of early modules.

Impact. The training effort is minimized; testing is enhanced; implementation time is reduced; and conversion is smoother. Individuals who helped define why a system was required are more likely to be present, which increases the likelihood of user satisfaction.

Reduced Chance Of Programming Team Turnover

By reducing the elapsed time prior to product delivery, the project leader reduces the amount of team turnover. On projects that could take several years (total time for the entire project) the Air Force assignment process induces a certain amount of team turnover. Using a modular approach could permit team members to work a module from start to finish, thus providing programming continuity.

Impact. Frustration levels triggered by training are reduced, proficiency is increased, and productivity is increased; while acquaintance and bonding with co-workers and user personnel can be maximized.

Reduced Chance Of User Management Turnover

Through accelerated product delivery, the software project leader is less likely to encounter new user management, and the increased possibility that they might want to make changes.

Impact. Project direction and philosophy changes are less likely to occur, will generally be smaller in scope and scale, and are usually less significant. Programming time and costs are kept on schedule and turmoil, as a result of change, is kept to a minimum.

Reduced Analysis Of Non-Implemented Requirements

The traditional approach usually requires that requirements be defined for all foreseeable system contingencies. In the course of the development process these are often altered enough to rule the original study effort worthless. In a large number of cases they are not implemented at all. The modular approach requires analyzing only those requirements necessary for the module being developed, with future modules incorporating changes or new options.

Impact. The defining of only those requirements to be implemented into system components will contribute to a higher level of worker productivity, will lead to reductions in both manpower and time, and will reduce system complexity.

Quickened Development Feedback To The User

Accelerating the rate at which some components of the system are delivered will allow the end user more time for adjustment. Since some components developed under the modular design approach could be delivered years ahead of those developed under the traditional approach, the user community will get hands on experience much earlier. Additionally, the user could have a direct impact on modules still being developed.

Impact. Partial system introduction, which comes about as a result of the modular approach, leads to a gradual user acceptance of the system and a general building of user confidence in the system being developed. The user only has to learn the system module being introduced; contrasted with the more or less bombshell approach of the traditional life cycle. Additionally, one of the the biggest drawbacks of the traditional approach is that the end user often has to wait until the very end of the development effort to discover the final product is not what he or she requires or that it does not do the things it was expected to do. The changes brought about as a result of these last minute corrections can be very expensive as well as time consuming. A final result of the modular approach could be the generation of greater user satisfaction.

SUMMARY

With the costs of computer software rapidly dominating the costs of military electronic systems, altering the traditional systems development life cycle may be one of the only viable options available to tomorrow's software programmers to increase productivity. The modular design approach will provide both tangible and intangible benefits to the Air Force, as well as a unique ability to take advantage of the dynamic environment encountered in high technology. The increased cost effectiveness and product quality of the modular design will ultimately lead to enhanced readiness for tomorrow's Air Force.

Chapter Six

CONCLUSIONS AND RECOMMENDATIONS

The "smart" systems of today's Air Force derive their IQs from their software. Software has proven to be crucial to a multitude of systems; from precision-guided munitions to the advanced combat aircraft and its fully integrated avionics suite. None can effectively operate without it, yet these systems and others like them may go into decline for future lack of good software and software professionals. Hardware may be readily available, but the software to exploit its superior capabilities may be lagging dangerously behind. The net result could be another system "outdated" before it is fully delivered and operational. As the range of computer applications has grown and the complexity of tasks increased, the cost of developing software has increased dramatically; so much so that software is in many cases the costliest component of the entire system. Almost every new electronic system the Air Force is developing is planning on software to help it function; yet, few people consider the efforts it takes to get that software. Some will be sorely disappointed at how long it might take.

In the early days of computer programming the usual assumption was that the work to be done was a simple product; a result of constant manpower over a scheduled period of time. Today, however, computer programming is extremely complex and more times than not is characterized by cost and time overruns. Unless productivity goes up and costs come down, there will be grave consequences for future Air Force capabilities. It is for these reasons the traditional systems development life cycle must be altered and a modular design incorporated. The remainder of this study consists of a summarization of those conclusions the author considers the most significant and some recommendations on how this modular (altered) design can be implemented in Air Force software programs.

CONCLUSIONS

1. Using an altered systems development life cycle, in this case a modular design, will dramatically reduce total systems development costs for Air Force software programs. Since studies on smaller programs using modular design revealed total lines of code could be reduced by 10-25 percent (with a 10 percent decrease yielding cost savings of up to 27 percent), it is felt total cost savings of 10-15 percent could be attained using a

modular design. Therefore, the hypothesis that altering the traditional systems development life cycle will provide economic benefits to the Air Force, is considered valid.

2. The modular design approach can be incorporated in a large number of Air Force software projects and programs. The reductions in cost and manpower can have a pronounced and timely effect on the successful accomplishment of many programs. It is possible the use of a modular design could permit some programs to be successfully completed when they might have otherwise been in jeopardy.

3. A major modification to one module does not render the entire program unusable; instead, portions of code can be moved in and out of the same source program with less difficulty than in non-modular programs.

4. Modular programming fosters modular testing, which inturn facilitates the testing of independent program modules prior to program completion. Discovering significant design flaws as early as possible and not during traditional system release is extremely crucial. The cost of rework is trivial in the early stages of development (as would occur under a modular approach) when compared with making changes late in the life cycle, as often occurs in the traditional life cycle.

5. Modular design offers many intangible benefits (such as reduced turnover, reduced analysis, and quicker feedback) which greatly enhance the productivity and quality of Air Force software programs. These types of improvements are hard to quantify, but may be just as important as the manpower, dollar and time savings which can be quantified.

RECOMMENDATIONS

1. The benefits of modular design should be shared with the higher levels of Air Force management, particularly those which oversee the development of Air Force software programs.

2. The Air Force should utilize a modular (altered) systems development life cycle wherever possible.

a. Current on-going programs should be evaluated for possible utilization of the modular approach.

b. Future programs should be evaluated for possible utilization of the modular approach; basing manpower and time estimates, as well as system costs on modular design parameters.

3. The Air Force should develop a training program to teach modular design. This training should be conducted for all programmers, software project managers, and systems analysts.

BIBLIOGRAPHY

A. REFERENCES CITED

Books

1. Cotterman, William W. (ed), et al. Systems Analysis and Design: A Foundation for the 1980s. New York: North Holland, Inc., 1981.

2. Fox, Joseph M. Software and Its Development. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1982.

3. Gujarati, Damodar. Basic Econometrics. New York: McGraw-Hill Book Company, 1978.

4. Larr, L., et al. Planning Guide for Computer Programming Development. Santa Monica, California: Systems Development Corporation, 1965.

5. Martin, James T. An Information Systems Manifesto. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1983.

6. Putnam, Lawrence H. Software Cost Estimating and Life-Cycle Control. New York: The Institute of Electrical and Electronics Engineers, Inc., 1980.

Articles and Periodicals

7. Boehm, B. W., et al. "Some Experience With Automated Aids to the Design of Large-Scale Reliable Software." IEEE Transactions on Software Engineering, Vol. SE-1, No.1 (March 1975), pp. 125-133.

8. Canan, James W. "The Software Crisis." Air Force Magazine, (May 1986), pp. 46-52.

9. Daly, Edmund. "Management of Software Development." IEEE Transactions on Software Engineering, Vol SE-3, (May 1977), pp. 230-242.

10. Latham, Donald C., and David R. Israel. "A Modular Building-Block Architecture." Signal, (May 1986), pp. 185-202.

CONTINUED

11. Snyders, J. "Slashing Software Maintenance Costs." Computer Decisions, Vol 11, No. 7, (July 1979), pp. 44-50.

Official Documents

12. AFR 700-1, Managing Air Force Information Systems, 2 March 1984.

13. AFR 700-4 (Volume I), Information Systems Program Management and Acquisition - Information Systems Program Management, 15 March 1985.

14. Department of Defense. Software Technology for Adaptable, Reliable Systems (STARS) Program Strategy. Washington, D.C., 15 March 1983.

15. General Accounting Office Report FGMSD-80-4. Contracting for Computer Software Development--Serious Problems Require Management Attention to Avoid Wasting Millions. Washington, D.C., Novemeber, 1979.

16. Mitre Corporation. Software Acquisition Management Guidebook. Beford, Massachusetts, 1978.

17. TRW Systems. A Software Acquisition Guidebook. Redondo Beach, California, Novemeber 1978.

Unpublished Materials

18. Hughlett, Eric C., Lieutenant Commander, USN. "A Framework for Software Development," Unpublished master's thesis, Naval Postgraduate School, Monterey, California, Septemeber, 1984.

19. Klemas, Gary H., Major, USAF. "Software Maintenance Cost Estimating." Research study prepared at the Air Command and Staff College, Report 83-1325, Air University (AU), Maxwell AFB, Alabama, 1983.

20. Norden, Peter V. "Project Life Cycle Modelling: Background and Application of the Life Cycle Curves." Working Papers of the Software Life Cycle Management Workshop, Airlie, Va., August 21-23, 1977, sponsored by US Army Computer Systems Command, pp. 217-306.

21. Swanson, E. B. "Software Maintenance." Proceedings of the Second International Conference on Software Engineering, UCLA, 1976, p. 492.

B. RELATED SOURCES

Books

Athey, Thomas H. Systematic Systems Approach, An Integrated Method for Solving Systems Problems. Englewood Cliffs, New Jersey: Prentic-Hall, Inc., 1982.

Brooks, Fredrick P., Jr. The Mythical Man-Month: Essays on Software Engineering. Reading, Massachusetts: Addison-Wesley Publishing Company, 1982.

McClure, Carma L. Managing Software Development and Maintenance. New York: Van Nostrand Reinhold Company, 1981.

Articles and Periodicals

DeRoze, Barry C. and Thomas H. Nyman. "The Software Life Cycle." Signal, (November-December 1977), pp. 5-8.

Fagan, M. E. "Inspecting Software Design and Code." Datamation, (October 1977), pp. 133-135.

Ingrassia, Frank S. "Combating the '90% Complete' Syndrome." Datamation, (November 1977), pp. 171-174.

Patrick, R. L. "Software Engineering and Life Cycle Planning." Datamation, (December 1976), pp. 21-27.

Rogers, Michael. "Software Makers Battle the Bugs." Fortune, (February 17, 1986), p.83.

CONTINUED

Official Documents

Air Force. Air Force Systems Command. A Review of Software Cost Estimation Methods. Hanscom AFB, Massachusetts, 1976.

Rand Corporation. Software Reliability: Philosophy Underpinnings. Santa Monica, California, 1974.

Unpublished Materials

Castle, Steve G., Maj, USAF. Software Reliability: Modelling Time-to-Error and Time-to-Fix. Unpublished master's thesis, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, Ohio, 1978.

END

7-87

DTIC