Logistics Management Institute

Maintenance of Department of Defense Mission Critical and Mission Support Software

A Preliminary Characterization

LG518T1

November 1997

Elizabeth K. Bailey
Emanuel R. Baker
James A. Forbes
Donald W. Hutcheson

LMI

Maintenance of Department of Defense Mission Critical and Mission Support Software

A Preliminary Characterization

LG518T1

November 1997

Elizabeth K. Bailey
Emanuel R. Baker
James A. Forbes
Donald W. Hutcheson

Prepared pursuant to Department of Defense Contract DASW01-95-C-0019. The views expressed here are those of the Logistics Management Institute at the time of issue but not necessarily those of the Department of Defense. Permission to quote or reproduce any part except for government purposes must be obtained from the Logistics Management Institute.

LOGISTICS MANAGEMENT INSTITUTE
2000 CORPORATE RIDGE
MCLEAN, VIRGINIA 22102-7805

Maintenance of Department of Defense Mission
Critical and Mission Support Software:
A Preliminary Characterization

LG518T1/NOVEMBER 1997

Executive Summary

Within DoD, mission critical software maintenance has been reported to cost between $700 million and $20 billion annually. The wide range results from uncertainty over the definitions of "mission critical" and "software maintenance," as well as the lack of any catalog of performing activities. The problem, however, is deeper than definitions and uncertainty over level of investment. The software maintenance process is poorly characterized in general, so there is no real basis for establishing coherent policy. Further, key software maintenance decisions—such as the choice of contract or organic performance and whether it should be defined as depot maintenance—are largely ad hoc and reap limited benefit from the results of past decisions.

The purposes of this study were to characterize DoD mission critical software maintenance in terms of its activities and processes, users and stakeholders, amount of resources, and existing formal and informal policy; identify policy issues; and outline the scope and major features of potential new or revised policy. This study was conducted at the direction of the Deputy Under Secretary of Defense (Logistics).

RESULTS

The terms "software maintenance" and "software support" are both in use, sometimes with modifiers such as "post-production" or "post-deployment." To avoid confusion, we adopted the term software maintenance and defined it as including

- correction of defects,
- adaptation (e.g., to a new host operating environment), and
- incremental functional improvements.

This definition is generally consistent with industry usage. Excluded from this definition are major modifications and upgrades, the purpose of which is major functional improvement.

We found it helpful to distinguish among three categories of mission-related software: mission critical, embedded; mission critical, nonembedded; and mission support. Broadly speaking, within a category different organizations may use similar processes; across categories they generally do not.

It is also helpful to characterize software maintenance by application area. We gathered data on six major applications: weapon systems; space control; automated test equipment (ATE); command, control, and communications; system integration laboratories; and simulation and training. Given the current state of data availability and reasonable limits on the study scope, it proved impractical to ensure completeness for any category or to achieve a reasonable degree of completeness for other than the first three.

FINDINGS

Within the six categories, we were able to account for about 16,000 government and contract personnel equivalents performing software maintenance, 55 percent organic and 45 percent contractor. The related total annual expenditure is about $1.26 billion annually. About 40 percent of the effort is corrective and 60 percent a combination of adaptive and incremental improvement. The code base that corresponds to these same categories is about 278 million source lines of code.

The use of operations and maintenance funds is almost universal for software maintenance. The amount of resources is normally determined as a level of effort rather than built up from discrete requirements. This approach appears to be consistent with industry software maintenance practice.

Software for the application areas studied is normally developed in the private sector. Although there were many transition patterns from original equipment manufacturer (OEM) to maintainer, three reasonably clear trends emerged:

- Pure organic maintenance is the exception for any type of software.
- Organic maintenance of embedded software is generally found only on older models of weapon systems.
- When attempted for nonembedded software, competitive contract support proved both more economical and at least as effective as either sole-source contract support or organic support.

There is a lack of consensus over what software maintenance is also depot maintenance. For this reason, inclusion or exclusion of software maintenance when reporting compliance with Title 10 U.S.C. limitations on depot maintenance outsourcing (the 60/40 rule) is not consistent across the department.

Written policy consists of military standards and local operating instructions rather than DoD instructions or service regulations. Not surprisingly, given the de facto status of the military standards as policy, their ongoing elimination was an issue for almost all of the organizations we interviewed.

RECOMMENDATIONS

We make two sets of recommendations, one set related to general policy and a second related to how DoD organizes for software maintenance.

Policy

Standardize on the term software maintenance, defining it to include correction of defects, adaptation, and incremental improvements. Exclude major modifications.

Define software maintenance in the weapon system, ATE, systems integration laboratory, and space control categories as depot maintenance. All four categories are either embedded in or closely tied to mission essential platforms.

Routinize consistent reporting of depot-level software maintenance, as defined above, in the AP-MP(A)-1397 *Depot Maintenance Cost System* report to provide a basis for reporting to Congress and management of depot-level software maintenance generally.

Invest in process improvement. Consider mandating minimum process capability levels for both organic and contract activities performing software maintenance.

Organizing for Software Maintenance

To achieve scale economies, consolidate smaller software maintenance activities into software maintenance centers of excellence. For each center of excellence, establish or keep a strong central management structure.

For embedded software, plan for long-term OEM maintenance. However, it is important to retain enough work organically to maintain a "smart buyer" capability.

For mission critical, nonembedded software, continue consolidation using the government-managed, contractor-performed, centralized maintenance model employed by the Army Communications Electronics Command and the Air Force Space Systems Support Group.

Where feasible combine development and maintenance within one organization. Where not feasible to do so, provide software maintenance organizations a greater voice in the definition of system requirements, particularly the development environment and documentation that will be delivered.

For software (such as ATE test program sets) where the software engineering knowledge is relatively easy to transfer, consider competition in order to reduce cost.

Contents

Figures

TABLES

Acknowledgments

The authors are indebted to the many Department of Defense software maintenance professionals who carved significant amounts of time out of their busy days to permit us to interview them for this report. All of the organizations we interviewed cooperated fully, were universally interested in this project, and had obviously devoted significant preparation time in advance of the interviews. We would especially like to thank the following:

- Mr. Dennis Turner and Mr. James Wagner of the Army Communications and Electronics Command (CECOM) Research and Development Engineering Center, Software Engineering Division, Fort Monmouth, NJ, who took great interest in the study, helped us frame the overall approach, and then followed up to see what other help they could provide.

- Mr. Mike Reed of the Air Force C4 Agency, Software Management Division, Software Process Improvement Branch at Scott AFB. Mr. Reed met with us twice, once to review our approach and once to critique preliminary results. His interest was sufficient that he met with us on a vacation day.

- Mr. Darrell Maxwell and Mr. Charles Bechtel of the F/A-18 Weapon System Support Activity at China Lake, CA, who were among the most articulate in describing the essentials of successful process improvement.

- All of the personnel whom we interviewed at the Air Force Consolidated Integration Support Facility (CISF) at Peterson Air Force Base, CO. We conducted seven interviews at the CISF; everybody was prepared in advance, candid in his or her assessments, and contributed a depth of insight that was vital to the final results.

- Mr. Waynard "Dev" Devers at the Institute for Defense Analysis who arranged for us to have access to software data originally collected in support of the Commission on Roles and Missions (CORM) of the Armed Forces. The CORM database is the foundation on which we built in creating a demographic picture of DoD software maintenance.

Chapter 1
Introduction

Study Background and Purpose

Within DoD, mission critical software maintenance has been variously reported to cost between $700 million and $20 billion annually. However, there has been no generally agreed-upon definition of what comprises mission critical. That being the case, there is no credible estimate of the resources involved in mission critical software maintenance. Similarly, there is no definitive list of performing activities, software maintenance that is also depot-level maintenance, processes used, formal and informal policy, or high- and low-cost drivers (or good and bad results). Lacking an adequate characterization of software maintenance, there is no real basis for establishing normative expectations regarding who should do it, how it should be managed, or how it should be funded. As a result, software maintenance decisions—such as contract or organic performance and levels of funding—are largely ad hoc, are difficult to reach, and reap limited benefit from an understanding of the results of past decisions.

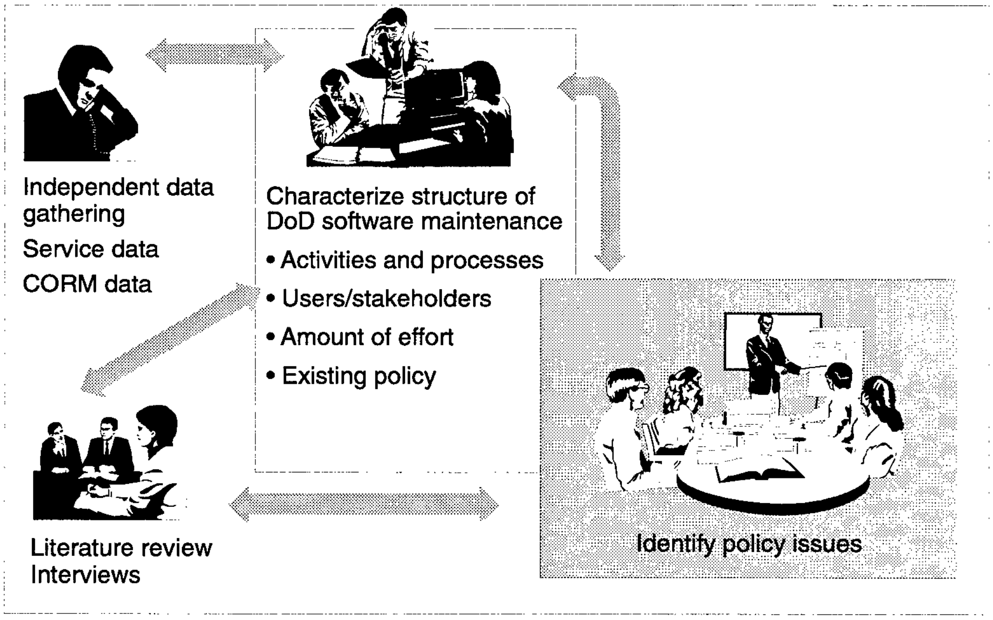
The purposes of this study were to characterize DoD mission critical software maintenance in terms of its activities and processes, users and stakeholders, amount of resources, and existing formal and informal policy; to identify policy issues; and to outline the scope and major features of potential new or revised policy. This study was conducted at the direction of the Deputy Under Secretary of Defense (Logistics).

Approach

Our study approach is illustrated in Figure 1-1.

In order to respond to the tasking, we separated the research into two segments, one quantitative and one qualitative. To establish the "demographics" of software maintenance (e.g., rough order of magnitude estimates of the code base, number of people performing, and annual cost), we started with a database created by the Institute for Defense Analysis for the Commission on Roles and Missions (CORM) of the Armed Forces. Because it was clear from the beginning that this database (the result of a data call to the services) had some voids, we supplemented it with data we obtained directly from the services. This study does not include software maintenance performed by defense agencies; the decision to exclude defense agencies was driven by the need to establish a reasonable scope of effort for what was envisioned as primarily an exploratory study.

Figure 1-1. Study Approach



In order to approach the more qualitative aspects, such as those having to do with the software maintenance process, we began with a literature review and then conducted a series of 15 semistructured interviews at 8 service installations. In keeping with the unsettled nature of software maintenance, we focused on developing an understanding of the common norms, meanings, values, and organizational relationships.[1] We were not so much trying to determine "facts" as discern signposts and perspectives.[2] In combination, the demographics research, literature review, and interviews permitted us to do this by characterizing software maintenance in terms of activities and processes, users and stakeholders, amount of effort, and existing formal and informal policy. Policy issues, in turn, flow from that characterization. The sites visited are shown in Appendix A, and the questionnaire used to conduct structured interviews is in Appendix B.

FINDINGS

The terms software maintenance and software support are both in use and their meanings are unclear. Sometimes these terms are used with modifiers, such as post-production or post-deployment, but generally without a sense of what either

[1] Kalle J. Lyytinen and Heintz K. Klein, "The Critical Theory of Jurgen Habermas as a Means for a Theory of Information Systems," *Research Methods in Information Systems*, ed. E. Mumfored et al. (Holland: Elsevier Science Publishers B.V., 1985) p. 221.

[2] U. Kelle, "Theory Building in Qualitative Research and Computer Programs for the Management of Textual Data," *Sociological Research Online*, Vol. 2, No. 2, http://www.socresearchonline.org.uk/socresonline/2/2/1.html. ¶3.9.

maintenance or support encompasses. To resolve this ambiguity, we adopted the term software maintenance and defined it as including

- correction of defects,
- adaptation (e.g., to a new host operating environment), and
- incremental functional improvements.

Excluded from this definition are major modifications and upgrades, the purpose of which is major functional improvement. Further, we defined software maintenance as beginning once a system has passed acceptance testing and has been delivered to the user.

Our use of the term software maintenance is generally consistent with the Institute of Electrical and Electronics Engineers (IEEE) definition; namely, it is "the process of modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a changed environment."[3] The IEEE also defines three subcategories of software maintenance:

> *Corrective maintenance.* Software maintenance performed to correct faults in hardware or software.
>
> *Adaptive maintenance.* Software maintenance performed to make a computer program usable in a changed environment.
>
> *Perfective maintenance.* Software maintenance performed to improve the performance, maintainability, or other attributes of a computer program.

The one difference between the definition we developed and that of the IEEE is the substitution of "incremental functional improvements" for "perfective." Our reason for this substitution is that none of the software professionals whom we interviewed was comfortable with the term "perfective" or used it. By contrast, "adaptive," "corrective," and "incremental improvement" were natural parts of their vocabulary. The definition we propose is also consistent with the way the work is actually being managed in DoD: the three activities of defect correction, adaptation, and incremental improvement are normally managed and performed in concert with one another as part of a single effort rather than correction of defects (maintenance in a classical hardware sense) being separate from adaptation and incremental functional improvements.

The term "mission critical software" is used as a catchall, sometimes as a synonym for "embedded software" (e.g., hosted in aircraft and tanks) and sometimes

[3] Institute of Electrical and Electronics Engineers, IEEE *Standard Computer Dictionary: Compilation of IEEE Standard Computer Glossaries*, New York, 18 January 1991, pp. 16, 55, 127, and 152.

defined more broadly. In analyzing the patterns of software maintenance, we found it helpful to be more specific and distinguish among three categories of mission-related software that undergo maintenance: mission critical, embedded; mission critical, nonembedded; and mission support (Table 1-1). Broadly speaking, within a category different organizations may use similar processes; across categories they generally do not.

Table 1-1. Software Maintenance Categories

Type	Cardinal characteristics	Examples
Mission critical, embedded	• Tightly coupled interfaces • Real-time response requirements • Very high reliability requirements (life critical) • Generally severe memory and throughput constraints • Often executed on special-purpose hardware	B-1 flight software, F-14 flight software
Mission critical, nonembedded	• Multiple interfaces with other systems • Constrained response time requirement • High reliability but not life critical • Generally executed on commercial off-the-shelf (COTS)	Command, control, and communications (C3), space systems
Mission support	• Relatively less complex • Self-contained or few interfaces • Less stringent reliability requirement	Automatic test equipment (ATE) Test Package Sets (TPSs), mission planning, business systems

These categories correspond roughly to those described in Boehm as embedded, semidetached, and organic.[4] Because in DoD maintenance the term "organic" is frequently used when referring to the government labor force, we have substituted the term "support" to describe the third class of software. Support software includes ATE and, more specifically, TPSs as well as software for simulation and training. The distinctions among these three classes of software have several important implications:

- ◆ They differ in their complexity and, consequently, in their cost to develop and maintain. [In fact, Boehm's Constructive Cost Model (COCOMO)—the most widely used software cost model—has three different cost and schedule equations to cover these three different types of software.] Embedded software is much more complex and costly to develop and

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

maintain. It is characterized by tightly coupled interfaces with hardware components and often with other hardware-software systems, real-time response requirements, very high reliability requirements, and often very constrained memory and processing capacity.

- They differ in terms of the nature of the associated maintenance activity, which primarily consists of functional enhancements for the first two and defect corrections for the third.
- They differ in terms of the skill set and amount of tacit knowledge required for maintenance—with implications for who can maintain the software, i.e., organic personnel or original equipment manufacturer (OEM) contractor.

Within the scope of the study, we accounted for an estimated 16,000 government and contract persons performing software maintenance on 278 million source lines of code (SLOC) at a cost of $1.26 billion annually. Given the current state of data availability and reasonable limits on the scope of this exploratory study, it proved impractical to develop census data on DoD software maintenance. We gathered data on the first six application areas in Figure 1-2, which depicts our subjective judgment of the degree to which each area was covered. The darkest shading means that the area is well covered, both in terms of on-site interviews of organizations maintaining that category of software and in terms of representation in our quantitative sources of data. The lighter shading means that the category has some representation in the study but also known major omissions. No shading means that the category is not represented in the study. Weapons systems, ATE, and space control systems are reasonably well covered in the study. C3, system integration, and simulation and training and are partially covered. The remaining categories are not covered.

Figure 1-2. Approximate Completeness of Data (Quantitative and Qualitative)

	Application area	Type	Data completeness
Data complete	Weapon systems	Embedded	Essentially complete
	Space control	Nonembedded	Essentially complete
	Automated test equipment	Support	Essentially complete
Data partially complete	C3	Nonembedded	Partial
	System integration	Nonembedded	Partial
	Simulation and training	Nonembedded	Partial
Data incomplete	Atmospheric search	Nonembedded	None
	War games and mission rehearsal	Nonembedded	None
	Nonembedded	None	
	Intelligence	Nonembedded	None
	Business systems	Support	None
	Weather	Nonembedded	None
	Other	----------	----------

For the six categories shaded in Figure 1-2, we were able to account for an estimated 16,000 government and contract personnel performing software maintenance, about 55 percent organic and 45 percent contractor. The related total annual expenditure is about $1.26 billion annually. About 40 percent of the effort is corrective and 60 percent is a combination of adaptive and incremental improvement. The code base that corresponds to these same categories is about 278 million SLOC.

Use of operations and maintenance (O&M) funds is almost universal for software maintenance within the application areas studied. The amount of resources is normally determined as a level of effort rather than built up from discrete requirements. In some organizations the level of effort was fixed in terms of dollars; in others it was fixed by the (fairly stable) size of the labor force. In either case, software maintainers addressed the backlog of requirements to the extent resources permitted. Requirements not satisfied in one planning period (e.g., year) were simply deferred to the following period. This approach also appears to be consistent with industry software maintenance practice.

There are three reasonably clear trends relating to choice of organic or commercial support. Software for the application areas studied is originally developed in the private sector. Although there were many transition patterns from OEM to maintainer, three reasonably clear trends emerged:

- Pure organic support is the exception for any type of software.
- For embedded software, there was a consistent pattern of early OEM maintenance lead followed by—only for older models of weapon systems—transition to organic lead. We believe that this pattern dominates for embedded software because the systems engineering knowledge needed to maintain it is difficult and costly to transfer.
- When attempted, competitive contract support proved both more economical and at least as effective as either sole-source contract support or organic support.

There is a lack of consensus over which software maintenance is also depot maintenance. The Defense Depot Maintenance Council *Business Plan for Fiscal Years 1996–2001* shows $275 million of contract depot-level software maintenance and 3.2 million direct labor hours of organic maintenance for FY96.[5] By contrast, the AP-MP(A)-1397 *Depot Maintenance Cost System* (DMCS) report, which explicitly requires reporting of depot-level software maintenance, shows $20.4 million for FY96. Our interviews with software maintenance managers confirmed the lack of consensus over what categories, if any, of software maintenance are also depot-level maintenance.

[5] DoD Defense Depot Maintenance Council, *Business Plan for Fiscal Years 1996–2001*, Tables 1-2 and 1-3, pp. 1-10 and 1-11.

Because of the lack of consensus, inclusion or exclusion of software maintenance when reporting compliance with Title 10 U.S.C. limitations on depot maintenance outsourcing (the 60/40 rule) is not consistent across the department.

Although there is not yet any general sense of what distinguishes effective and ineffective software maintenance, there are candidate criteria. Certain organizations we visited had a better sense of what they did, and were better able to explain it, than others. On reflecting on what seems in some sense to be consistent patterns of behavior across these organizations the following characteristics stand out:

- They take commitments seriously and are able to follow through on them.
- They are able to articulate organizational objectives for improvement and to follow through with actions to reach them.
- They participate throughout the system life cycle, not just after deployment.
- They make effective use of contractor support, competing contracts when that makes sense and fostering productive long-term relationships among sole-source providers at other times.
- They have the necessary quantity of people and resources along with strong, central leadership within the organization.

Of the organizations we visited, the F/A-18 program stands out as having all of these characteristics. The F/A-18 program provides an especially interesting example because it moved from a self-acknowledged near-disaster to a state of health in less than 5 years. However, all three services had some organizations with some if not all of these characteristics. Also worth noting, and of more than passing importance, is that some of the premier mission critical software is maintained by organizations that do not satisfy the above criteria.

Written policy consists of military standards and local operating instructions rather than DoD instructions or service regulations. Not surprisingly, given the de facto status of the military standards as policy, their elimination was an issue for almost all of the organizations we interviewed.

RECOMMENDATIONS

We make two sets of recommendations, one set related to general policy and a second related to how DoD organizes for software maintenance.

Policy

Standardize on the term software maintenance in lieu of alternatives, such as software support. Define software maintenance to include correction of defects, adaptation, and incremental improvements. Exclude major modifications.

Define software maintenance in the weapon system, ATE, systems integration laboratory, and space control categories as depot maintenance. All four categories are either embedded in or closely tied to mission essential platforms.

Routinize consistent reporting of depot-level software maintenance, as defined above, in the AP-MP(A)-1397 Depot Maintenance Cost System report to provide a basis for reporting to Congress and management of depot-level software maintenance generally. In the absence of a compelling reason for broader reporting, we recommend against expanding beyond depot-level software maintenance. Because of both the current lack of consensus on basic definitions and the many different organizational hierarchies involved, it would be costly to mount and sustain this type of effort.

Invest in process improvement. Specifically, we recommend mandating minimum process capability levels for both organic and contract activities performing software maintenance. The mandated capability levels should be judiciously expanded over time.

Organizing for Software Maintenance

Consolidate smaller software maintenance activities into software maintenance centers. Size each center such that it has an annual business base of approximately $100 million or greater. For each center of excellence, establish or keep a strong central management structure.

For software embedded in a single weapon system platform, recognize that long-term OEM software maintenance is a given and plan for it. However, it is also necessary to retain enough work organically to maintain a smart buyer capability.

For mission critical, nonembedded software, continue consolidation using the government-managed, contractor-performed, centralized maintenance model employed by the Army Communications Electronics Command (CECOM) and the Air Force Space Systems Support Group (SSSG).

Where feasible, follow the F/A-18 model and combine development and maintenance under one organizational umbrella. Where not feasible to do so, provide software maintenance organizations a greater voice in the definition of system requirements such as what development environment and what documentation will be delivered.

For support software (such as ATE TPSs) where the software engineering knowledge is relatively easy to transfer, consider converting to essentially 100 percent competed contract performance in order to reduce costs.

Chapter 2
Software Maintenance Demographics

INTRODUCTION

The purpose of this chapter is to convey the relative size of DoD mission critical and mission support software maintenance activity. To do so, we present data on three key demographics: source lines of code (SLOC) supported, number of people involved, and estimated budget impact.

As noted in Chapter 1, there are wide discrepancies in the estimates of dollars spent per year on the maintenance of mission critical software. At the high end is a $20 billion figure, which MITRE derived by estimating that $30 billion is spent by DoD annually on software and that two-thirds of that is for maintenance.[1] At the low end is an estimate of $700 million derived from the results of a 1994 Commission on Roles and Missions (CORM) data call to the services.[2]

Both of these estimates have significant methodological difficulties. It is unclear in the MITRE analysis whether the $20 billion total is for all DoD software, mission critical only, or some other subset of the DoD total. The CORM result contained known areas of omission in addition to terminological confusion. Thus, an important objective of the present study was to establish the magnitude of software maintenance with greater confidence. In addition to characterizing the magnitude of software maintenance in terms of dollars, we also looked at the size of the code base being maintained and the number of personnel (government and contractor) maintaining this code base.

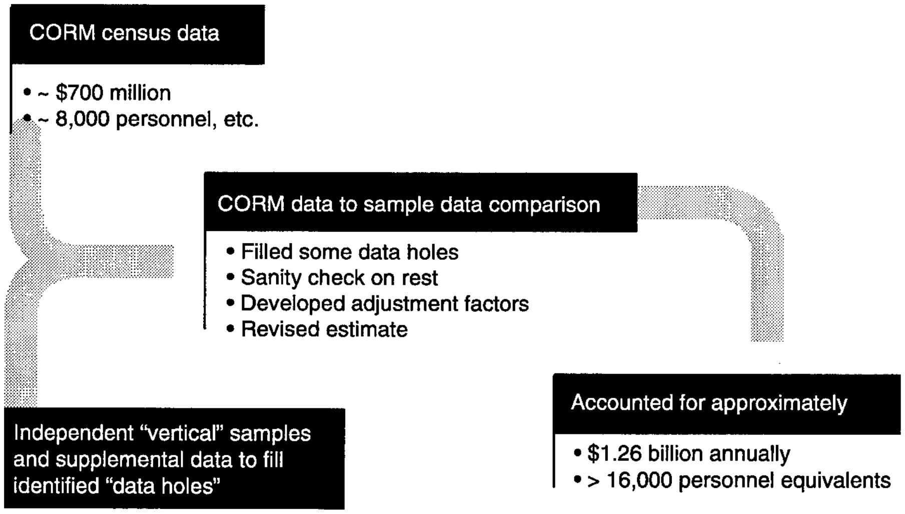
The CORM database was used as a starting point since it was recent, bottom-up, and based on a formal data call. We filled in fairly obvious gaps (such as space control systems) that were missed entirely by the CORM data call and asked subject matter experts in all three services to identify and correct other errors (e.g., significant underrepresentation of ATE software). In addition to enlarging the CORM database, we refined it by comparing the CORM data for a given site with data obtained from our site visits. We looked in particular for any systematic under- or over-estimates in the CORM data on size (SLOC) and people.

The overall process we used to quantitatively characterize software maintenance is illustrated in Figure 2-1.

[1] Barry M. Horowitz, *The Importance of Architecture in DoD Software*, MITRE M91-35, 1995, p. 2-3.

[2] Computed by members of the IDA research staff from service submissions in response to the CORM data call.

Figure 2-1. Mission Critical Software Magnitude Estimate Process



CORM census data

- ~ $700 million
- ~ 8,000 personnel, etc.

CORM data to sample data comparison

- Filled some data holes
- Sanity check on rest
- Developed adjustment factors
- Revised estimate

Independent "vertical" samples and supplemental data to fill identified "data holes"

Accounted for approximately

- $1.26 billion annually
- > 16,000 personnel equivalents

CODE BASE

We used SLOC to estimate the size of the code base being maintained. This was not without reservations. The SLOC metric suffers from inherent ambiguities. Some software engineers count physical lines while others count instructions. Additional sources of ambiguity are whether or not comments and nonexecutable lines, such as data declarations, are counted. In spite of these known problems, SLOC is the most common measure of software size.[3] Other size measures have been proposed, the most notable being function points.[4] However, the function point method is not in wide use for embedded weapon system software and, for this reason alone, could not be used in the current study.

As noted earlier, we used site visits to check on the validity of the CORM SLOC counts. We asked the interviewees to verify the counts we had, making any corrections or additions. We also looked for evidence of any systematic over- or under-estimation across the various sites. While there were corrections and additions, there were no systematic differences, and, with a few major exceptions, the CORM numbers appeared to be relatively accurate for the sites where we were able to do a comparison. The CORM database, however, is only a partial database. For example, it omitted the Air Force's space control software entirely and missed most of the ATE TPS software, so the site visits were also used to fill in missing data, but this is at best a partial and limited enlargement of the CORM database.

[3] Barry W. Boehm, *Software Engineering Economics*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1981, pp. 479, 482.

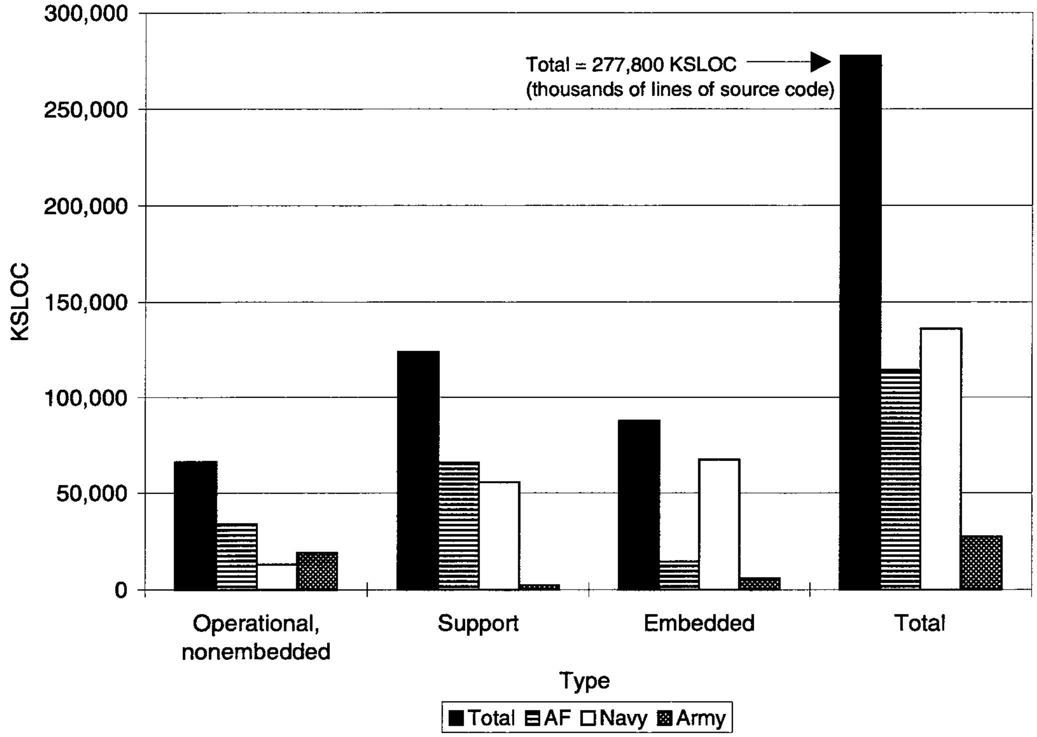
[4] International Function Points Users Group, *Function Points Counting Practices Manual,* Westerville, OH, 1994.

As discussed in Chapter 1, we defined three general categories of mission critical software: embedded, nonembedded, and support. In reality, these categories represent points on a continuum rather that clearly discrete classes. It is especially difficult in some cases to classify systems as embedded or nonembedded. We used the following guidelines in classifying systems:

- We classified as *embedded* anything integral to the operation and performance of an aircraft, ship, missile, gun, etc.
- We classified as *nonembedded* satellite control software and command and control software.
- We classified as *support* TPSs, trainers, and simulators.

Figure 2-2 shows a breakout by the three high-level categories for each service. The Navy and Air Force have much larger code bases than does the Army.

Figure 2-2. Software Code Base by Service and Category



While support software is the single largest category in terms of the sheer number of SLOC, it is less costly to maintain than the other two categories. As an indicator of the size of the difference, Table 2-1 reflects the approximate cost per source line of code per year for three of the sites in the expanded database.

Table 2-1. Representative Maintenance Costs by Category

Category	Approximate maintenance cost per line of code per year
Embedded	$110.00
Nonembedded	$5.60
Mission support	$0.81

Note: The mission support cost is calculated from North Island ATE TPSs, nonembedded is calculated from CECOM data, and embedded is calculated from B-1B data.

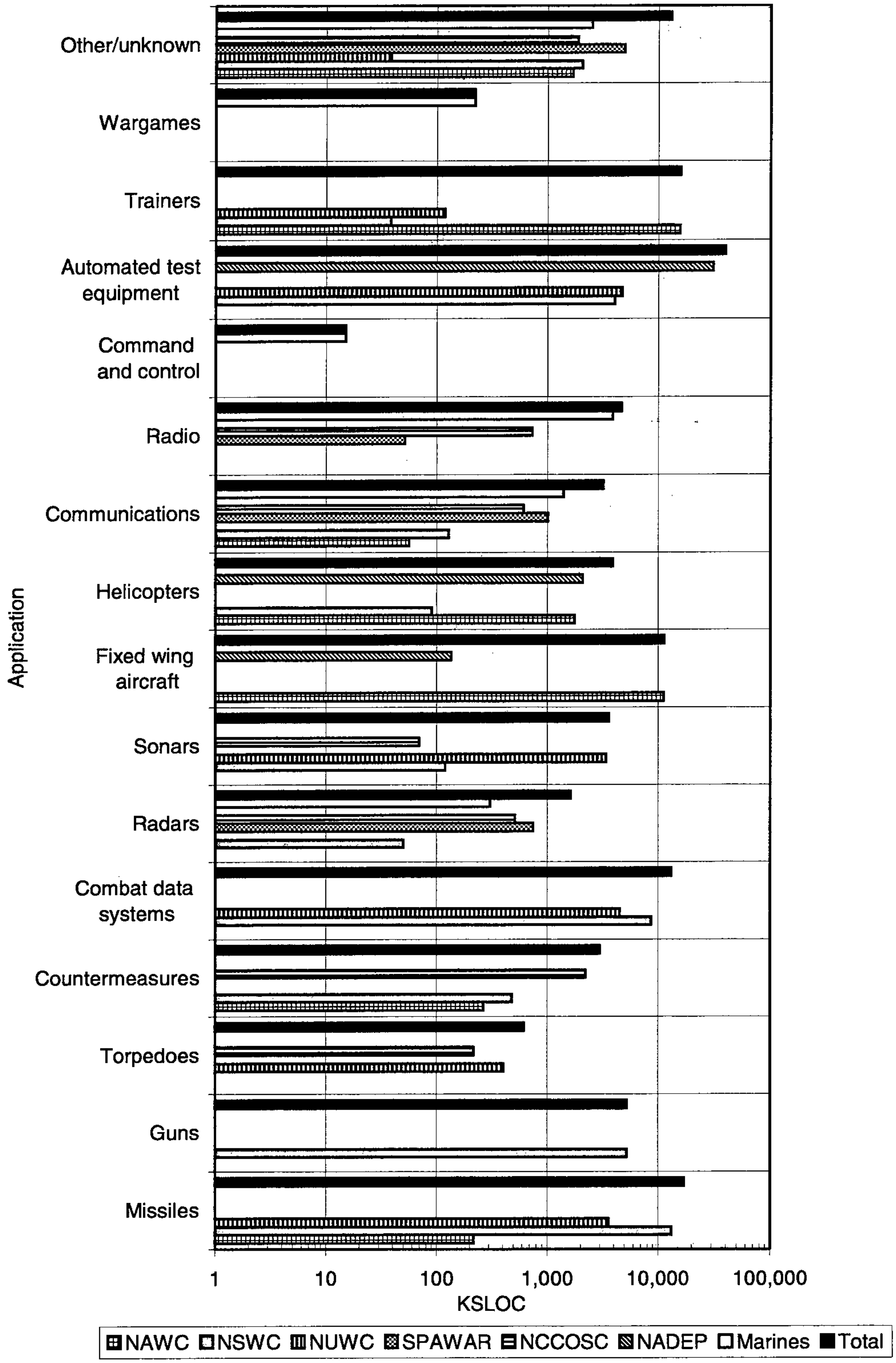
In addition to the overall volume of code being maintained in DoD, the amount of code by application area is also of interest. Figures 2-3 through 2-5 show the number of source lines of code by application area and organization for each service.

Figure 2-3 shows a breakdown of SLOC by organization and application for the Navy. Of 136 million SLOC reported by the Navy, the warfare centers are responsible for maintaining over 90 million. The Naval Aviation Depots (NADEPs) are responsible for 33 million, of which 31 million are for support software (ATE). The Marines reported a total of 8 million.

Figure 2-4 shows the same breakdown for the Air Force. Of the 114 million SLOC reported, 87 million are maintained by the air logistics centers, primarily Ogden, Oklahoma City, and Warner-Robins. The SSSG at Colorado Springs is responsible for the balance. As was the case with the Navy, support software is the single largest category, containing 60 million SLOC.

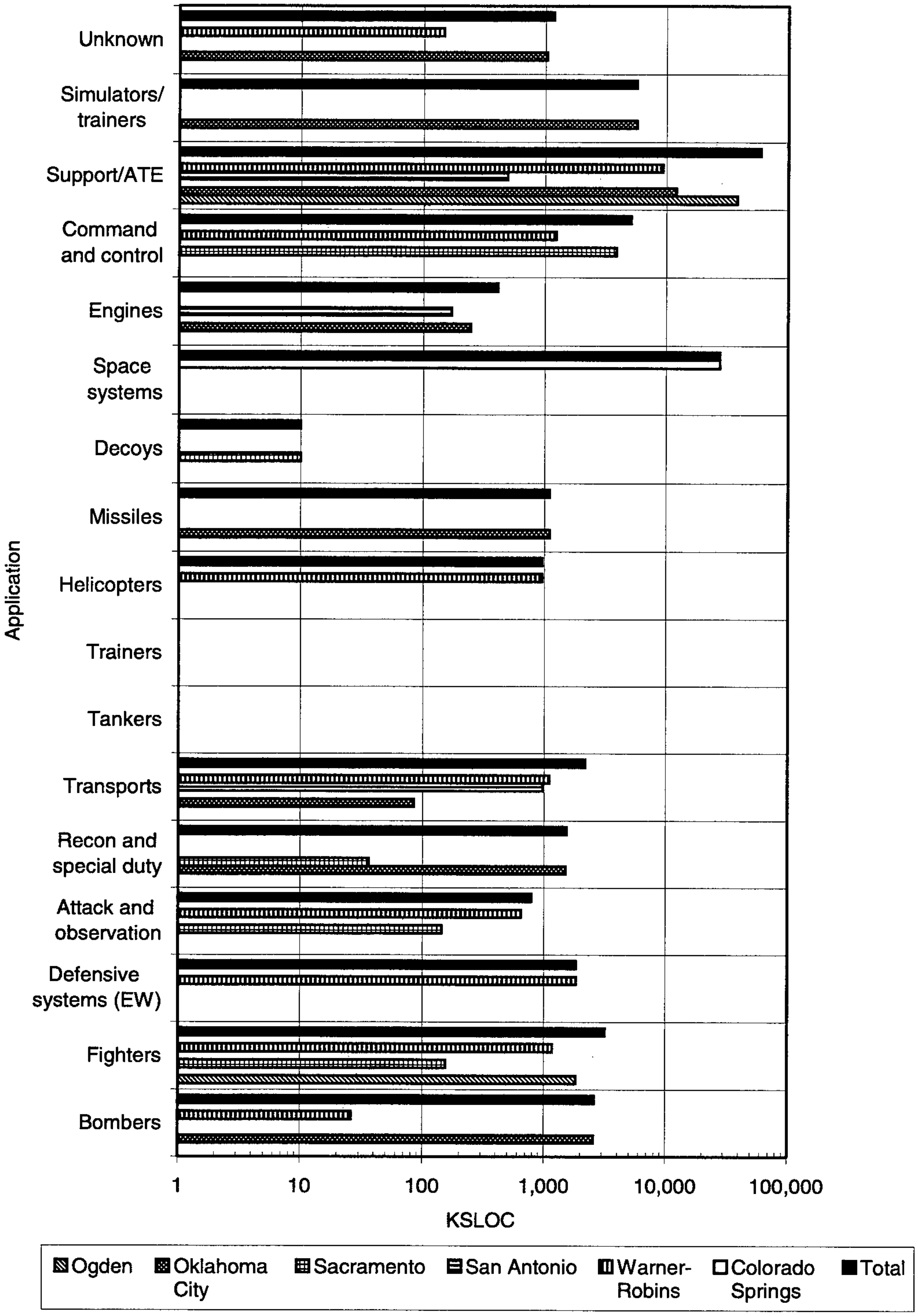
In interpreting Figures 2-2 through 2-4 it should be remembered, as noted earlier, that there are significant reliability and validity issues with the underlying data. Although our check of code counts reported in the CORM database against those made available in site visits did not reveal a systematic bias, that is not the same as saying the data are known to be valid. Because only three of the six application areas we examined were, in our judgment, reasonably complete, this summary is an underestimate even for the areas we examined. The portrayals shown here are best characterized as approximate representations of the relative sizes of the code bases for the categories we examined. These caveats also apply to the labor force and budget impact demographics presented later in this chapter.

Figure 2-3. Navy Source Code Counts by Application and Organization



Note: Total = 114,450 KSLOC (thousands of lines of source code).

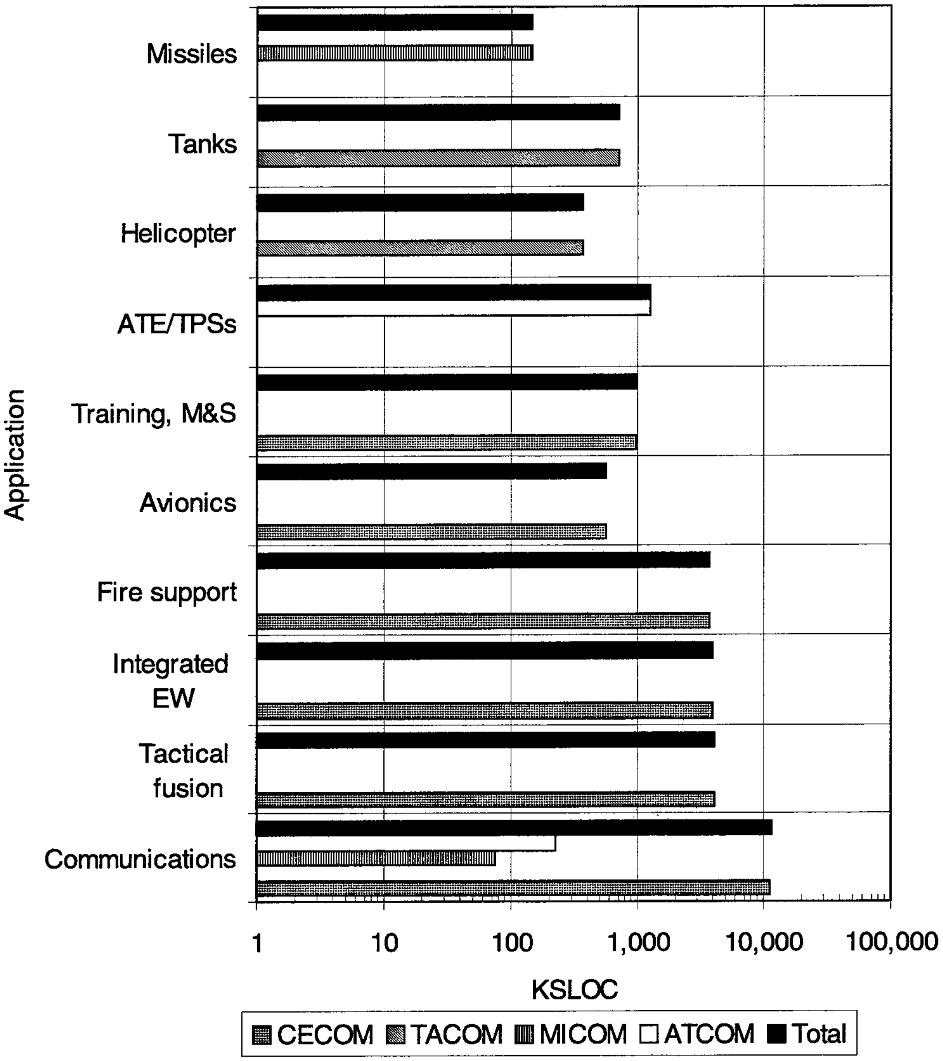
Figure 2-4. Air Force Code Base by Organization and Application



Note: Total = 136,050 KSLOC (thousands of lines of source code).

Figure 2-5 shows the same breakdown for the Army. Of the 27 million total, almost 25 million are maintained by CECOM, 11 million of which are for communications.

Figure 2-5. Army Code Base by Organization and Application



Note: Total = 27,350 KSLOC (thousands of lines of source code).
M&S = modeling and simulation.

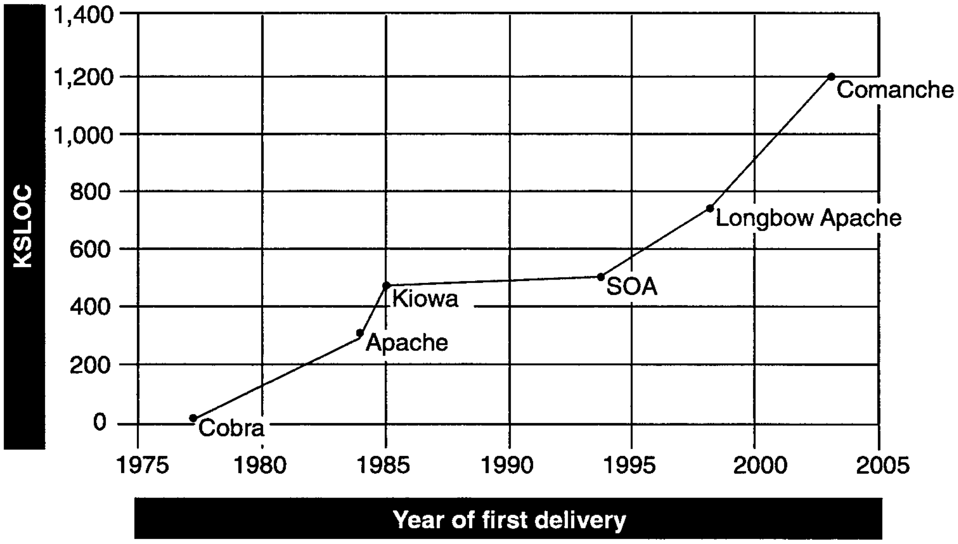
These code counts represent a horizontal snapshot across services, platforms, and application areas. The longitudinal change over time is also important. Table 2-2 shows the amount of software code in three generations of strategic bombers.[5] Figure 2-6 does the same thing over a succession of helicopter platforms.

Table 2-2. Source Code by Bomber Type, Model and Series

Bomber type, model, series	Year first aircraft delivered to the Air Force	KSLOC
B-52H	1981[a]	100
B-1B	1985	500
B-2A	1993	1,800

[a] Although the B-52H was introduced in 1961, the offensive avionics were converted from analog to digital in 1981, a more reasonable milestone for this purpose.

Figure 2-6. Source Code for Selected Helicopter Platforms



Note: Code counts courtesy of Army Aviation and Troop Command. Years of first delivery from *Janes' All the Worlds Aircraft*, 1995–1996; *Air Force Magazine*, June 1996; and Army Factbook at http://www.dtic.mil/armylink/factfile/comanche.html.

The trends in Table 2-2 and Figure 2-6 confirm the general sense of the interviewees that software density is increasing with succeeding generations of weapon systems.

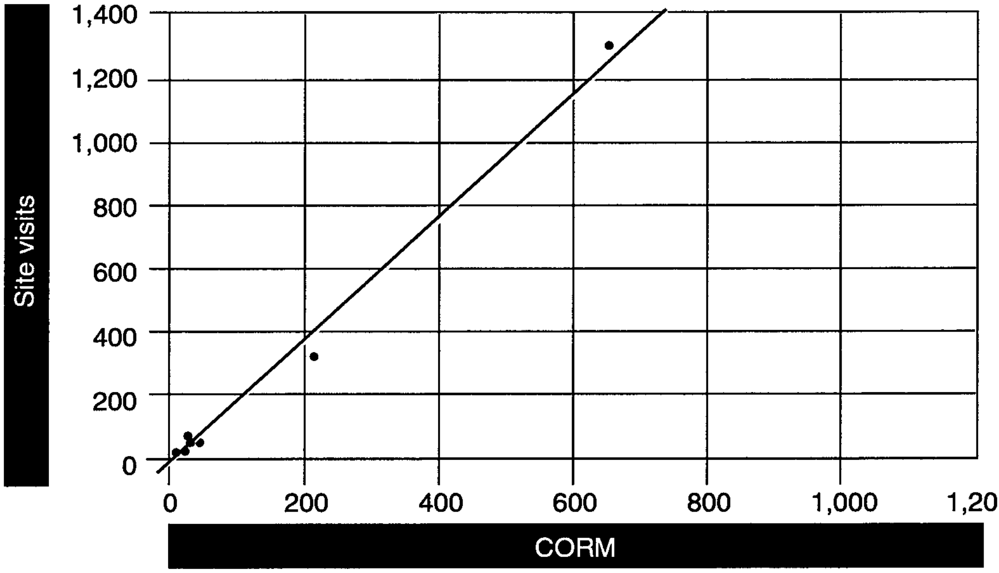
[5] Years of introduction from United States Air Force fact sheets at world-wide web URL http://www.af.mil/news/factsheets, 7 April 1997. Code counts from George Koleszar, et al., *FY95 Heavy Bomber Force Study*, Institute for Defense Analysis Report R-394, July 1995 and George Koleszar et al., *Cost and Operational Effectiveness Analysis (COEA) for the B-1B Conventional Mission Upgrade Program (CMUP)*, Institute for Defense Analysis Report R-398, Draft Final, November 1996.

PERSONNEL

Software development and maintenance are labor-intensive activities. In fact, human effort is generally recognized to be the major cost driver.[6,7] To estimate the number of people involved in software maintenance, we began with the CORM database personnel counts. Here also we expanded the CORM database using other data gathered during the study. As we did with the size data, to determine accuracy we compared the numbers obtained from the site visits with those in the CORM database.

The CORM database consistently underrepresented the number of people. A comparison between the CORM and site visits is shown in Figure 2-7. If the data from the site visits and the CORM data for the same sites were about the same, then a linear plot of the data would have a 1:1 slope. The slope is 1.96, meaning that the personnel counts obtained from the site visits were almost twice as large as those from the CORM data call, and this was consistent for all but one of the sites we visited. The one outlier was the F/A-18 aircraft. The CORM data call reflects 30 F/A-18 personnel, all organic, while interviews with F/A-18 software managers indicate the total should be approximately 1,000 (125 organic plus 875 contractors). Since it was such an egregious outlier, we did not include the F/A-18 in calculating the 1.96:1 site-visit-to-CORM data ratio.

Figure 2-7. Personnel Data from Eight Site Visits Compared to CORM Data for Same Sites

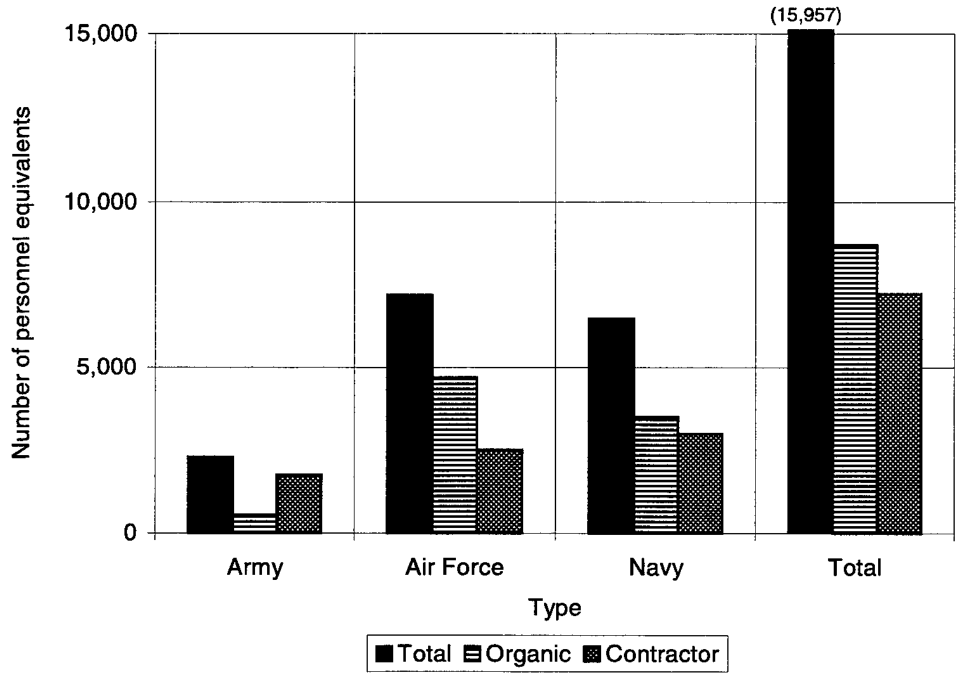


[6] Boehm, *Software Engineering Economics.*

[7] Wolfhart B. Goethert, Elizabeth K. Bailey, and Mary B. Busby, *Software Effort and Schedule Measurement: A Framework for Counting Staff-Hours and Reporting Schedule Information,* CMU/SEI-92-TR-21, ESC-TR-92-021, Carnegie Mellon University Software Engineering Institute, Pittsburgh, PA, September 1992.

For the specific sites for which we had data, the corrected counts were used. We adjusted all other numbers in the CORM database by 1.96, essentially doubling the CORM counts. Figure 2-8 shows the breakdown of organic and contractor personnel for each of the services. Overall, there are almost 16,000 people maintaining software for the mission critical and mission support applications included in this study. Overall, this labor force is about 55 percent organic, although the ratio varies by service. The Navy and Air Force rely primarily on organic personnel, whereas the Army has fewer organic personnel than contract.

Figure 2-8. Organic vs. Contractor Personnel for Each Service



Budget Impact

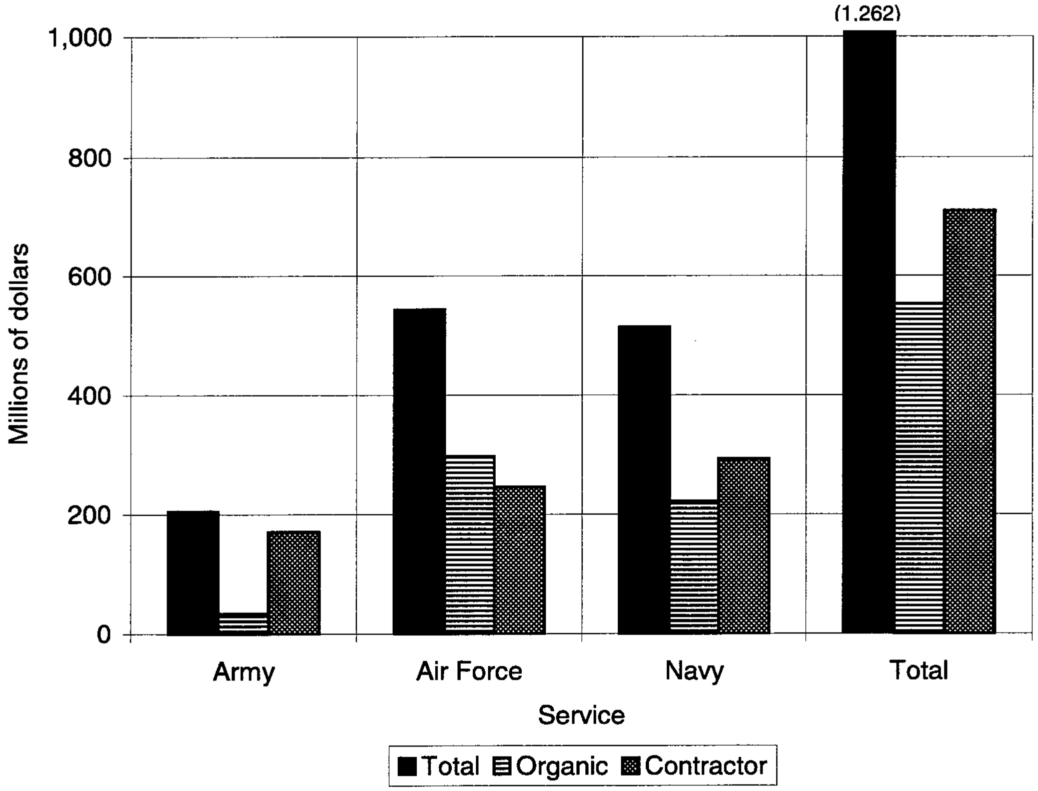
The third measure of magnitude is financial (dollars). We did not use the budget numbers from the CORM data call because it is not clear what these reflect (i.e., labor only or labor and equipment, contract or contract plus organic, etc.). As an alternative, we estimated the financial commitment in dollars by multiplying counts of people by average loaded labor rates for organic and contractor personnel. Figure 2-9 shows the estimated dollars per year for each service.

The rate used for organic personnel was $67,364, which is a composite rate based on an assumed distribution of 80 percent GS-12 and 20 percent GS-13 (1996 dollars).[8] The rate used for contractor personnel was $97,364, which is the median of

[8] The composite organic rate is a weighted average of the rates shown for GS-12s and GS-13s in Table A26-1 of the *Civilian Standard Composite Pay Rates by Grade*, Air Force I instruction 65-503, May 1996.

the rates that were quoted to us during the site visits. The contractor rates ranged from $55,500 to $250,000 per year, and this difference generally corresponded with the complexity and uniqueness of the software being maintained. The difference between organic and contractor rates should not be interpreted to mean that contractors are more expensive. By and large, the contractor labor force was maintaining more complex software that required higher skills.

Figure 2-9. Estimated Budget Impact by Service



The financial commitment that we were able to account for using this procedure is approximately $1.26 billion dollars annually ($205 million for the Army, $543 million for the Air Force, and $514 million for the Navy).

As noted in Chapter 1, one of the reasons for characterizing DoD software maintenance was to shed light on the amount of software maintenance that is also depot-level maintenance. Whether software maintenance is or is not depot level is of interest because it affects the department's compliance with the congressional restrictions on how much depot maintenance work can be outsourced.[9]

It is not possible at present to describe what fraction of the $1.26 billion in software maintenance is depot level. First, it was clear from the interviews that, here also, there is a lack of consensus over definitions. For example, the Air Force would generally classify work on fighter aircraft embedded software as depot maintenance. The Navy does not consider it so. Hence, inclusion or exclusion of software maintenance when reporting compliance with Title 10 U.S.C. limitations on depot maintenance outsourcing (the 60/40 rule) is inconsistent. Uncertainty in this area is quite large. The *Defense Depot Maintenance Council Business Plan for FY96–01*, which is compiled with service inputs, shows $275.3 million in contract depot-level software maintenance for FY96 and an additional 3.2 million depot labor hours worth of organic support. By contrast, the AP-MP(A)-1397 Depot Maintenance Cost System (DMCS) report, under which depot-level software maintenance is explicitly required to be reported, reflects $20.4 million for the same year.

[9] 10 U.S.C. 2466, "Limitations on the Performance of Depot-Level Maintenance of Material," requires that not more than 40 percent of the funds available in a fiscal year to a military department or agency for depot-level maintenance and repair may be used to contract for performance by non-federal-government personnel.

Chapter 3
Software Maintenance Processes

INTRODUCTION

This chapter describes the software maintenance processes in terms of certain qualitative characteristics. The most relevant characteristics, which were selected after a review of available literature and exploratory interviews, are

- the typical patterns for transitioning from the original equipment manufacturer (OEM) who designed the software to the steady-state maintenance organization,
- the process by which maintenance requirements are communicated from the user to the maintainer,
- the typical software maintenance version release cycles,
- the software maintenance budgeting process,
- the equipment and facilities used for software maintenance,
- the entity outside of the maintenance organization who monitors software maintenance performance,
- the software maintenance process improvement efforts,
- training,
- the metrics used to measure and manage software maintenance,
- the funding sources for process improvement and capital investments,
- the operable software maintenance policy,
- the advances and initiatives,
- the lessons learned as reported by maintenance organizations,
- the changes desired by maintenance organizations, and
- the characteristics that appear to be associated with effective software maintenance organizations.

In the following sections, we address each of these characteristics in turn.

MAINTENANCE RESPONSIBILITY TRANSITION PATTERNS

Because a major reason for this study was understanding the basis for selecting contract or organic software maintenance, we asked the interviewees during our site visits to describe the basis for whatever arrangement they had in place. With the exception of one program office at the SSSG, none of the interviewees were able to articulate why they had the arrangements they did, nor did we find much help in the literature. To the extent that there was any response from interviewees other than at SSSG, the answer typically was that some organic maintenance is necessary to preserve smart buyer capability.

Since explanations for the present arrangements were not available, it was necessary to infer first the categories of arrangements and then infer why these arrangements existed.[1] We did so by determining who was currently providing maintenance and the pattern for transitioning from the OEM developer (in all cases we studied software was developed under contract) to the present maintainer. Figure 3-1 illustrates nine transition patterns that we either found or that appeared reasonable to anticipate. We found examples for seven of the nine.

These patterns had the following characteristics:

- Pattern I is OEM development followed by OEM sole-source maintenance. We found one example of this pattern, the Global Positioning System Operational Control System (GPS OCS).
- Pattern II is OEM development followed by pure organic support. We found only one example of this pattern, maintenance of ATE TPSs at the Warner-Robins Air Logistics Center (WR-ALC).
- Pattern III is OEM development followed immediately by competed commercial maintenance (without the OEM as one of the competitors). We did not find any representatives of this pattern.
- Pattern IV is OEM development followed by joint OEM/organic maintenance followed by a transition to competed contract maintenance. The Defense Support Program software followed this pattern.

[1] John W. Creswell. *Research Design: Qualitative and Quantitative Approaches.* Thousand Oaks CA: Sage, 1994, pp. 153–161. The method we used is sometimes called grounded theory—i.e., grounding a theory in the data.

Figure 3-1. Maintenance Responsibility Transition Patterns

Pattern type	Design	Maintenance provider							Workloads
		Pure OEM (sole source)	"Pure organic"	OEM	Completed (OEM 3rd party)	Organic	Organic	Lead	
		—	—	Organic	—	OEM	Completed (OEM 3rd party)	Support	
I	OEM	▶							SDG (GPS OCS)
II			▶						WR-ALC ATE
III		▷			▶				
IV				▷	▶				SDB (DSP)
V				▷		▶			F-14, F/A-18, F-15 WR-ALC EW
VI					▶				CECOM, SDN, SMS
VII						▶			
VIII							▶		No. island avionics No. island ATE
IX						▷	▶		SDD
User software maintenance appears to be relatively rare									

▷ = Transitional ▶ = Final

Note: SDG (GPS OCS) = Global Positioning System Operational Control System; WR-ALC ATE = Automated Test Equipment at Warner Robins Air Logistics Center; SDB (DSP) = Defense Support Program; SDN = Air Force Satellite Control Network; SMS = Space Defense Operations Center; SDD = Defense Meteorological Satellite Program.

- Pattern V is OEM development followed by joint OEM and organic maintenance. Typically, the OEM has the lead at first (e.g., as each new series of aircraft is introduced), then organic personnel take the lead as the software ages. This pattern is typical for embedded software in combat aircraft. It appears that this pattern has emerged for embedded software because the detailed system knowledge needed for embedded software maintenance is very hard to transfer from one organization to another.

- Pattern VI is OEM development followed by a transition to competed commercial support when the OEM is one of the competitors, but not necessarily the winner. CECOM uses this model, as do two of the programs at SSSG in Colorado Springs.

- Pattern VII differs from Pattern V in that there is no intermediate, OEM-lead stage. We did not find a representative of this pattern.

- Pattern VIII has the organic sector leading the maintenance effort but with competed contract support. North Island Naval Aviation Depot uses this model for maintenance of avionics software and TPSs.

- In Pattern IX, the organic sector always leads the maintenance effort. Early on, the OEM provides support; later, the support role is competed. We found one space systems program using this model.

Analysis of these patterns and comments offered during the interviews suggest the following provisional conclusions:

- Pure organic software maintenance is the exception and seems limited to mission support software, such as ATE TPSs. Since the organic and contract sectors have roughly the same skills and would be expected to use the same software environments, we conclude that, except for support software such as ATE TPSs, there is significant difficulty and cost associated with transferring the knowledge of the software necessary for its maintenance. In addition to problems with nondelivery of documentation or computer-aided software engineering environments, this knowledge is probably tacit (i.e., deep knowledge) rather than explicit—that is what makes it hard to transfer. The contrast between the simple TPS software typical of Pattern II and the complex embedded software typical of Pattern V supports this conclusion. Support for this conclusion is also found in the literature on technology management in which Teece,[2] examining how companies arrive at make-or-buy decisions, noted that they often choose what is easy to do rather than what is most important to them.

- Based on the empirical evidence (i.e., the established transition patterns) and the reasons presented under Pattern V, planning for pure organic maintenance or competed maintenance of embedded software is unrealistic. It is probably more realistic to accept OEM involvement in (and initial lead of) embedded software maintenance as a *fait accompli.*

- Competed commercial maintenance is viable for mission critical, nonembedded and for mission critical, support software.

[2] David J. Teece, "Technological Change and the Nature of the Firm," *Technological Change and Economic Theory*, 1988, pp. 256–281.

COMMUNICATION OF REQUIREMENTS

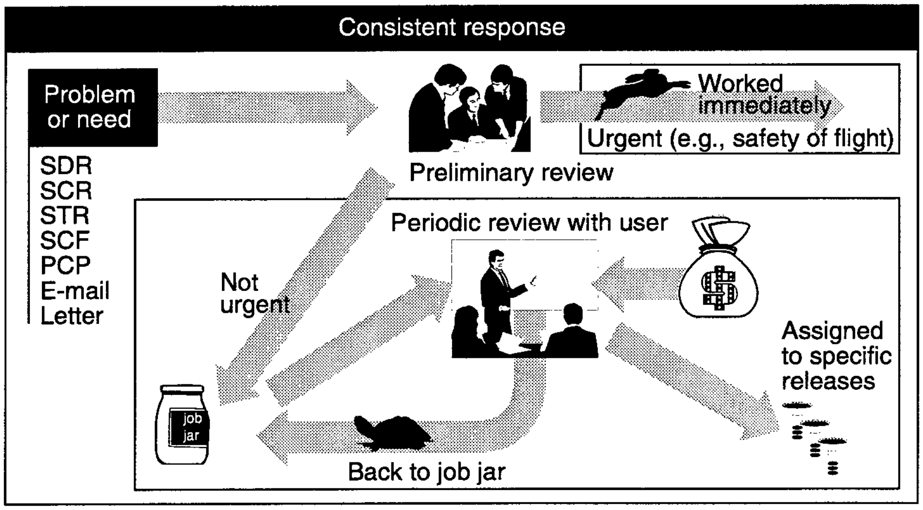
Communication of requirements is clearly an important part of the software maintenance process. We found uniformity in this process among organizations in the field survey. The typical requirements process (Figure 3-2) follows these steps:

- It is initiated by a user through a problem report or a request for change. These reports or requests had almost as many different names and acronyms as organizations surveyed. The names included system deficiency report (SDR), standard change form (SCF), software trouble report (STR), and program change proposal, or they could take the form of an electronic mail or letter input. (Interestingly, in the Air Force, no one reported using formal Technical order 00-35-D54 deficiency reports, even though this technical order applies to all Air Force agencies and organizations and provides for software deficiency reporting.)[3]

- The requests are typically screened in a preliminary review to determine the urgency of the problem or change request. Urgent needs (e.g., safety of flight) are worked immediately. The remainder of the requests are accumulated in what the Space and Warning Systems Directorate (accurately, if colloquially) termed a "job jar" awaiting a scheduled review.[4]

Figure 3-2. Requirements Process

Question: How are changes and requirements communicated and changes initiated?



Note: SDR = Software deficiency report; SCR = Software change request; STR = Software trouble report; SCF = Software change form; PCP = Program change proposal; E-mail = Electronic mail.

[3] United States Air Force, TO 00-35D-54, *USAF Deficiency Reporting and Investigation System*, 15 January 1994, pp. 1-1, 1-4, and 1-5.

[4] Space and Warning Systems Directorate, Operating Instruction 33-7, *Software Maintenance—Acronyms and Terms*, Vol. 2, 15 September 1995, p. 6. The term job jar is actually used in this operating instruction.

- The requests are reviewed by an established group (e.g., F/A-18 System Change Review Board) periodically. Prior to the review, initial estimates of the magnitude of the effort, which changes can be efficiently grouped, etc., are accomplished by an engineering staff. The reviews often (but not always) have user participation or inputs. The group chartered to do the review examines the requests in the job jar, prioritizes them, and selects software changes to be implemented. Selection is based primarily on priority and available funding.

- Requests not selected go back to the job jar for future consideration. Typically, there are more requests than funds.

- Problem reports or change requests selected for implementation are assigned to a software version release.

Neither the size of the backlog of requirements nor the specifics of particular requirements in the backlog drives the budget. Rather, planned support takes the form of a level of effort expressed in terms of either dollars or work force. Essentially, the agreed upon level of effort establishes a "cut line." On a prioritized list of software maintenance requirements, software changes above the line are implemented and those below it are deferred to the job jar for future funding opportunities. This behavior would indicate that most software maintenance tasks are not of a time-critical nature. It is worth noting that level-of-effort funding is found in commercial software maintenance practices.[5] (There are at least anecdotal indications that it is also found in commercial software development.)

SOFTWARE VERSION RELEASE CYCLES

Once the software change is approved for design, the process of changing the software begins. Typically (except for TPSs), software maintenance occurs in cycles called block releases. The number of software changes in a block release, the length of individual cycles, the time between version releases, and the structure and content of the process vary by category of software (i.e., embedded, nonembedded, support) and within categories. Since the greatest variation seemed to be between categories, we characterize the release cycles by category in the text that follows. The software release cycles seen were primarily the result of the nature of the systems being supported. The diversity seen in release cycles for nonembedded and support software was also a function of the diversity in the systems and needs.

[5] Alain Abran and Hong Nguyenkim, "Analysis of Maintenance Work Categories Through Work Measurement," *Proceedings of the 1991 IEEE Conference on Software Maintenance*, Sorrento, Italy, p. 105.

Mission Critical, Embedded Software

The most formal release process observed was for embedded software. The embedded system software change process typically follows a structured waterfall of required activities accomplished on an agreed upon schedule. The waterfall of activities includes engineering design, coding, testing, rework, and formal acceptance. Embedded systems normally have multiple changes in each software release. The typical duration of the embedded software change process is 1.5 to 3 years. Releases overlap in time, resulting in software releases to the field approximately every 2 years.

The length of the embedded software cycle and the overlapping cycles are a balance between the economy of scope for the costly testing necessitated by a highly integrated weapon system and the user's need to field changes in a timely manner. A third or more of the change process may involve regression testing to ensure proper function and to determine and eliminate any detrimental effects on other subsystems. For example, the F/A-18 followed a 36-month release cycle. Each cycle had 8 months of development testing and 5 months of formal validation and verification testing for each release. Thus, grouping of multiple changes in a software version release is the norm to make efficient use of the testing investment required for each cycle.

Mission Critical, Nonembedded

A variety of software change processes was observed across mission critical, nonembedded software maintenance in our field surveys. We observed everything from highly formalized, regularly scheduled releases to single changes made as needed. The change process is consistent over time for a given system but varied greatly across systems both in process duration and release intervals. This was true even of systems managed at the same organization. The patterns seen appear to be affected by the number of fielded systems and number of interfaces. The frequency of releases ranges from no software releases in a year to 15 releases per year.

Mission Support

For ATE TPSs, the only mission support software for which we have interview data, changes are primarily made in reaction to weapon system hardware changes or to correct initial defects. Changes are designed, tested, and fielded one by one rather than being grouped.

BUDGETING

Budgeting for software maintenance is almost universally in O&M funds. An exception was fielded Army systems still in production, where software maintenance is funded with production funds. Major software modifications for mission changes or substantial performance changes are budgeted in R&D funding and fall outside our definition of software maintenance.

Only one organization interviewed reported that funding was based on a specific need set and entailed a contract with the next level of authority to accomplish the specific efforts for the funds provided. More generally, software maintenance funding was not budgeted for specific requirements but was established to support a level of effort expressed in terms of dollars or work force. The level of funding was typically a negotiation between the software maintenance activity and the next level of authority.

There are at least three reasons for level-of-effort funding. First, a consistent level of effort makes it easier to keep a trained and responsive work force. One organization with whom we spoke made this case. Second, software maintenance requirements, unlike hardware maintenance requirements, are insensitive to operations tempo. Increased flying hours result in increased hardware maintenance costs due to reliability failures and wear. On the other hand, increased flying hours do not necessarily drive software costs in that software does not fatigue or wear. Third, software maintenance requirements are individually of such limited scope and impact that it is just not worth the overhead that would be required to manage them discretely. We suggest the second and third reasons without any real proof, however, since none of the organizations we visited articulated a reason (other than maintaining a stable work force) for managing to a level of effort. Level-of-effort funding or staffing has interesting implications for ability to reduce maintenance costs: improving maintenance productivity will have no effect on costs if it simply results in reaching deeper into the maintenance job jar. Reduction in maintenance costs will depend on simultaneous control over productivity and demand.

Air Force software maintenance organizations expressed budget process concerns unique to that service. Generally speaking, the Air Force has more rigid rules regarding the use of different types of funding and a more fine grained approach, with a number of different categories of O&M funds. As expressed by Air Force software maintenance managers, the funding rules add complexity and time to the software maintenance process. One software manager reported that 20 percent of his time was spent on funding issues. Software maintenance managers in other services did not report similar problems.

EQUIPMENT AND FACILITIES

We found a variety of software maintenance environments during the survey. These were generally a cumulative legacy of software development programs. Operating under such as legacy was not a problem when only one weapon system was supported at a facility—as was typical of embedded software. However, mission critical, nonembedded and support maintenance organizations that supported multiple systems expressed concern over the plethora of environments at their software maintenance facilities. Such diversity limited their ability to move people from support of one system to another. Not surprisingly, they expressed a need for more voice in the selection of the software maintenance environment during system development. Some of the organizations with whom we spoke stated that a common support environment (e.g., language or support processor) would benefit future systems, but also recognized that changing the current support environment was impractical.

The practicality of a common support environment is worthy of further discussion. Specifying common *target* environments is practical and is done today (e.g., the MIL-STD-1750 architecture for avionics computers). Under these circumstances, the contractor may develop the software on its own development environment and use a cross-compiler to produce the software that executes in the target environment. Specifying a common *support* environment means specifying a common *development* environment, since software maintenance is typically done on a carbon copy of the development host computer (and its associated software tools). Specifying a common support environment would mean that only a handful of computer environments could be made common, as a practical matter, and many companies would have to redo their development environments to meet the common standard. This would be costly and anti-competitive, and might even cause some companies to forgo defense software business.

Upgrades to equipment and facilities are usually funded by weapon system programs case by case. Support organizations surveyed typically had little if any of their own funds for upgrades. Only one organization amortized facilities to produce funds for facility upgrades. However, the organizations in general did not see equipment and facilities upgrades as a significant concern.

OUTSIDE MONITORING

The interviewees were asked "What outside sources monitor your software support activity?" We asked this question to identify people or organizations outside of the immediate software maintenance activity that collected or received information and had authority to influence what the activity does. We looked at who these outside organizations are, what it is they are monitoring, and whether there were any issues or problems in this area.

Monitoring took several forms. All organizations reported cost and schedule status on a periodic basis to the next higher level in their organizational hierarchy. Depending on the service and the source of funding, these are program management activities (PMAs), program executive offices, program managers (PMs), or system managers. Another form of monitoring is through operational test, which is required for major weapon systems (e.g., F-14, F/A-18) and subsystems (e.g., defensive systems). Operational tests are conducted by user representatives. We also found cases of peer group monitoring in both the Navy and Air Force, but peer groups, not surprisingly, did not appear to have veto power over the software maintenance decisions. Rather, they made recommendations and promulgated best practices. Beyond the three forms of monitoring described, the organizations we interviewed were largely self-managed.

From the perspective of those we interviewed, there are no issues or problems raised with respect to monitoring. In the words of one Navy manager, "We are responsible for meeting schedules and implementing functions and we're held accountable for that. Why do we need someone else?" However, from the perspective of headquarters organizations and OSD, there is a need for both quantitative and qualitative monitoring of these organizations to better characterize software maintenance. It is the general lack of this type of information that was the impetus for the current study.

PROCESS IMPROVEMENT

Process improvement (sometimes called business process reengineering) has taken center stage within DoD and the commercial world as a primary means to realize productivity and quality gains. For this reason, we asked the organizations we visited about their software maintenance process improvement efforts.

Alternative Frameworks

For software, there are two frameworks for guiding process improvements, the Software Engineering Institute (SEI) Capability Maturity Model (CMM) and the International Organization for Standardization (ISO) 9000 standard. Fundamental to both is the idea that an organization's software processes must first be assessed regarding baseline areas of strength and weakness. Improvements are then made relative to this baseline.

Based on experience of one of the authors (Baker), who has extensive experience in this area, the most widely followed framework for software process assessment and improvement within the United States is the CMM. The CMM comprises five maturity levels (Table 3-1). At each level there is a set of key process areas (KPAs), which are the focus of process improvement efforts at that level.

Table 3-1. Capability Maturity Model

Level	Characteristics	Example KPAs
1. Initial	• Lack of defined processes for project management or software engineering • Performance dependent on individual, rather than organizational, capability	N/A
2. Repeatable	• Basic project management controls in place	• Requirements management • Project planning
3. Defined	• Standard process across the organization • Software engineering process group facilitating process improvement	• Peer reviews • Organization process definition
4. Managed	• Quantitative quality goals for software products • Implementing corrective action based on process measures	• Software quality management • Quantitative process management
5. Optimizing	• Continuous process improvement • Rigorous causal analysis of defects and defect prevention	• Defect prevention • Process change management

The alternative is the ISO 9000-series quality standards, used as a basis for process assessment in concert with ISO's Software Process Improvement and Capability determination (SPICE), which is used as a basis for both software process assessment and improvement. The ISO framework is predominant in Europe and other areas outside the United States. ISO 9000 is a quality standard that is not specific to software but applies to any design and manufacturing process (e.g., automobile design and production). ISO has published ISO 9000-3, which contains a set of guidelines for applying ISO 9000 to software.[6]

Unlike the CMM, ISO 9000 has only one "level"—an organization is either registered or not registered. To obtain ISO 9000 registration, processes must be defined, documented, and followed. Specific processes are not mandated; instead, an organization must "say what they do and do what they say." This contrasts with the CMM, which is much more prescriptive in defining what has to be in place for the KPAs at each level. Also unlike the CMM, ISO's SPICE provides a continuous model for improvement rather than containing discrete levels. SPICE also introduces new methodological issues. With the CMM, Level "n" organizations can be compared to each other because they are evaluated using the same criteria. At Level 2, for instance, they have all achieved satisfaction of the same KPAs. With SPICE, however, organizations can pick and choose which KPAs to be evaluated

[6] Gianluigi Caldiera, "Impact of ISO 9000 on Software Maintenance," *Proceedings of the IEEE 1993 Conference on Software Maintenance*, Montréal, Quebec, Canada, pp. 228–230.

against. Consequently, to compare two SPICE Level 2 organizations, for example, one would have to know which KPAs they were evaluated against.

Among the sites we interviewed, the CMM was the framework of choice, with one exception. That exception was an ATE organization working toward ISO 9000 registration—which they chose specifically because it is not specific to software but applies to hardware as well. It is worth noting that this organization described what they were doing as test engineering, not software engineering.

The Grassroots Level

In our site visits, we found a consistent grassroots desire for process improvement. A major stumbling block, however, is obtaining the resources required. Process improvement takes a commitment of time, effort, and money. Resources are needed for training, defining and documenting processes, and formal assessments by people authorized by the SEI or ISO. In the experience of one of the authors (Baker) of this report, who is authorized by the SEI to lead assessments, a typical assessment can cost on the order of $50,000 to $60,000 (including the dollar equivalents for the time that participants spend in supporting the assessment process). We found only one organization that was able to command and keep the necessary resources to implement a long-term program of process improvement: the F/A-18 program. They reported that the move from a CMM Level 1 to a Level 2 took them 5 years, and then additional years to reach Level 5. While it required a long-term commitment, they felt that it clearly improved quality and schedule performance. This is not atypical: the SEI and others have published data showing clear improvements in software quality and productivity with increasing process maturity.[7]

We found other sites that had embarked upon programs of process improvement but expressed the concern that the decision-makers would pull the funding. In the words of one Air Force software manager: "I'm a firm believer in the CMM and hope that our leaders do not become impatient and stop funding software agencies' abilities to reach higher CMM levels." In fact, a software manager in another Air Force organization went on to tell us that in his organization funding for process improvement was cut by a two-star flag officer because progress was so slow. His organization kept working on it underground. Finally, there were organizations who wanted to improve their processes but did not begin to have the resources. For example, one Navy software manager had a total budget of $12,000 per year for all training for an organization of 55 personnel.

[7] J. Herbsleb and D. Goldenson, "A Systematic Survey of CMM Experience and Results," *International Conference on Software Engineering*, 1996.

Higher Organizational Levels

In contrast to the grassroots level, it was not clear how consistent the support for process improvements is at higher levels of DoD.

In the early 1990s, the Deputy Assistant Secretary of the Air Force for Communications, Computers, and Support Systems promulgated a formal directive mandating that all organic software activities (central design, software design, and software support) undergo a formal CMM assessment by October 1994. These assessments were carried out by personnel from the Air Force C4 Agency, Software Management Division, Software Process Improvement Branch at Scott Air Force Base. According to the directive, all organizations were to reach Level 2 by 1996 and Level 3 by 1998.[8] The Air Force organizations we interviewed at Colorado Springs and WR-ALC had been assessed and had active process improvement efforts underway. That directive has now been discontinued, and the future of software process improvement in the Air Force is unclear.

In May 1996, the Army mandated that their software organizations establish a process improvement program with the goal of reaching CMM Level 3 within 6 years of an initial process assessment.[9] In addition, the Army policy mandated that the process maturity of contractors be considered in source selection for software development and maintenance. The Army policy stated that the "SEI CMM... concepts and methodologies are widely used and internationally accepted in industry and DoD for defining and appraising the software process capability of an organization." The Army policy "strongly recommends" the use of the CMM in evaluating the process maturity of a contractor but allows for alternative frameworks, specifically ISO's SPICE.

The Navy had no formal policy related to software process improvements. Consequently, improvement efforts depended on the interest of software maintenance organizations and their ability to acquire the resources required for assessments and improvement activities. We found a wide range of process improvement efforts—everything from the F/A-18 organization, which had undertaken a long-term process improvement effort, to several Navy organizations that had no resources to embark on any kind of a process improvement program.

[8] Deputy Assistant Secretary of the Air Force for Communications, Computers, and Support Systems, Action Memorandum, Subject, *Policy on Software Maturity Assessment Program*, 23 September 1991.

[9] Department of the Army, HQDA LTR 25-96-3, "Software Process Improvement Policy," 24 May 1996.

TRAINING

Software maintenance, whether organic or contractor provided, requires a skilled labor force. When it is performed by contractors, the government still must maintain the skills to monitor the performance. During the interviews, we asked questions about

- the amount and nature of training,
- the percentage of the total software maintenance budget that was spent on training, and
- the prevalence of project-specific training over more generalized training in software engineering and new technologies.

Most respondents said that project-specific training was paid for by project funds and that general training was paid for out of overhead funds. In other aspects, the answers differed by service.

In the Air Force, both the SSSG at Peterson Air Force Base and the WR-ALC had a combined organic and contractor work force and both described ongoing training programs for organic personnel. The SSSG had a contract with an outside firm to provide training (both project specific and general). Managers at WR-ALC reported a fair amount of training activity (CMM, project specific, and general). The estimates of the percentage of their total budget spent for training ranged from 5 to 10 percent.

Managers at the Navy sites described a Naval Air Systems Command (NAVAIR) requirement of 40 hours per year per person for training. Taken literally, this would be about 2 percent of the labor budget (i.e., 40 hours per year divided by 2080 hours). The reported number of actual number of hours spent on generalized training ranged from 13 to 40 at the different sites. All sites reported project-specific training as well. The Naval Undersea Warfare Center reported a fair amount of variability—from essentially no training in certain divisions to fairly extensive training in general and project-specific training (representing a 10 percent commitment of budget) in others.

Neither of the two Army sites we visited described an ongoing training program. The software managed by both sites is almost entirely contractor maintained. They did report that, in some cases, contractors had funded their own training to improve their CMM rating.

In general, training for the organic or contractor labor force did not appear to be an issue to those we interviewed, with the exception of training needed for process improvement. Both Navy and Air Force organizations mentioned training for process improvement as a problem.

Virtually everyone we talked to pointed to the need for a well trained organic force to serve as an intelligent customer. A particular point of emphasis was the training needed by those in acquisition to ensure that the products required for maintenance are specified during system development. Symptoms of problems that were cited include the lack of adequate tailoring of DoD-STD-2167A to strike the right balance between too little and too much documentation as well as the failure to contractually require delivery of source code.

METRICS

During the interviews, we asked "What information (i.e., metrics, management indicators, etc.) do you use to monitor software support?" One would expect programs to track cost and schedule (dollars budgeted versus expended and milestones planned versus completed). We were interested in looking beyond these basic status indicators to the use of specific quantitative information about the software maintenance process or the product.

We placed the responses into one of the following three categories:

- The organization did not have a metrics program and did not track quantitative information beyond dollars and milestones.
- The organization reported metrics to some outside organization or group (usually as a result of a specific service policy).
- The organization used quantitative information to make internal management decisions and could readily describe or show us specific examples.

While the second and third categories are not, conceptually, mutually exclusive, in our sample of software maintenance organizations they appeared to be in practice.

Table 3-2 summarizes the responses. In interpreting this table, it is helpful to understand the policy related to metrics within each of the services. The Army required reporting of a total of 12 metrics reflecting such characteristics as requirements volatility, control-flow complexity, and completeness of testing. The purpose of these metrics is to indicate readiness for operational testing. They are known as the software test and evaluation panel (STEP) metrics and were mandated by the Army on 4 January 1993 for all programs that had not reached milestone II by that date.[10] As part of the Army policy, the STEP metrics were reviewed as part of the operational test readiness review.

[10] U.S. Army Director of Information Systems for Command, Control, Communications, and Computers Memorandum, "Preparation for Implementing Army Software Test and Evaluation Panel (STEP) Metrics Recommendations," 4 January 1993.

Table 3-2. Use of Metrics by Sites Interviewed

Organization	Basic status metrics	Metrics reported outside organization	Metrics used to make decisions
Army			
Communications Electronics Command (CECOM)		●	
Aviation and Troop Command (ATCOM)	●		
Air Force			
Colorado Springs		●	
Warner-Robins		●	
Navy			
North Island–ATE			●
North Island–Helicopters		●	
F-14			●
F/A-18			●

CECOM not only reported the STEP metrics but has produced a guidebook that relates the STEP metrics to specific program issues—*The Streamlined Integrated Software Metrics Approach (SISMA) Guidebook: Application of STEP Metrics.*[11] The STEP metrics were not required for the Apache helicopter because it was beyond milestone II when the Army policy was implemented. (We should point out that we spoke to the government organization responsible for the Apache helicopter. The software is maintained by McDonnell-Douglas in Meza, AZ. We did not speak with the contractor about any metrics they may have had for internal use.)

Air Force policy mandated the collection of five core metrics (size, effort, schedule, defects, and rework).[12] Although this policy has since been rescinded, at the time of our site visits, it was still in effect and was being followed by the Air Force organizations we interviewed. In addition, both Air Force organizations participated in a metrics working group, which was organized to share lessons learned across the Air Force Materiel Command.

The Navy had no service-wide policy related to software metrics. There was, however, a NAVAIR metrics working group, which met to share lessons learned

[11] Department of the Army Communications Electronics Command, Research and Engineering Center, Software Engineering Directorate. *The Streamlined Integrated Software Metrics Approach (SISMA) Guidebook: Application of STEP Metrics.* Ft. Monmouth, NJ, July 1993.

[12] Deputy Assistant Secretary of the Air Force for Communication, Computers, and Support Systems, "Software Metrics Policy," 93M-017, 16 February 1994.

related to metrics. The organization at North Island responsible for maintaining the Navy helicopter software did not participate in that group. The remaining Navy sites all used metrics to make specific management decisions. For example, the F/A-18 program used defects found during testing—categorized by priority—to decide when the software was ready to be fielded. The ATE organization at North Island described their use of historical data to predict the number of problem reports from the field that they will be required to investigate and, of those, the number that would require corrections (i.e., real problems not user error). These predictions were used to determine the budgetary requirements for the following year.

In general, and despite the fact that the Navy did not have a formal metrics policy, it was the Navy offices that were most likely to use measurement as an integral basis for management. It is not clear if this apparent trend is real or a product of our sampling.

FUNDING SOURCES FOR PROCESS IMPROVEMENT AND CAPITAL INVESTMENTS

All organizations had some form of support environment (e.g., computer hardware, networks, system simulators, compilers, and other support software) in place. Generally, these support environments were inherited from the acquisition programs under which the software was initially developed. Interviewees identified three types of funding for capital improvements of these support environments. They were

- other new systems (or major modifications to existing systems) in development,
- capital investment budgets, and
- depreciation on existing capital plant.

Of the sites visited, only CECOM identified a capital investment budget, and only WR-ALC indicated they depreciated existing capital plant to generate replenishment funds. All other organizations stated they had one route to upgrade: they depended on acquisition organizations to transfer the development environments associated with the new systems to them.

We did not note any particular concern with a lack of funding to upgrade environments. Of more concern—but this was not consistently articulated—was the heterogeneity of the support environments. Interviewees brought this problem up in the context of their not having enough influence over the acquisition process. (See the section on changes desired by maintenance providers for a more complete discussion of this perceived problem.)

Operable Policy and Military Standards

A primary reason for this study was to understand what is needed in the area of software policy. Consequently, this is a topic we explored in some detail during the interviews. Policy can be viewed from two different perspectives. First, it can be considered as representing required behavior (i.e., as formal, normative policy), the common view. Another perspective is to consider policy as providing a framework of consistent expectations regarding how players mutually interact (i.e., as facilitating cooperative action).[13] Given the relative absence of normative software maintenance policy, both perspectives were potentially important. Both formal and informal policy are covered in this report; this section focuses primarily on formal policy.

In order to explore policy issues, we asked the following question: "As you perceive it, what is the operable policy affecting software maintenance?" The interviewees, consistent with the previous discussion, were encouraged to answer this as broadly as possible and were told that "operable policy" referred to both formal and informal policy. We explained that "formal policy" refers to DoD, service, and organizational directives that mandate how something is to be done, while "informal policy" refers to generally understood ways of doing business. This question also triggered a great deal of discussion about policy-related concerns.

Policies Cited

The most frequently cited documents were several military standards that prescribed software engineering processes. Almost universally, DoD-STD-2167 or DoD-STD-2167A were mentioned. Several respondents listed MIL-STD-498 as well.[14] Two sites mentioned MIL-STD-1679. These military standards describe the documentation to be delivered, formal reviews to be held, and tasks to be addressed in developing or maintaining software. A fairly broad variety of other documents were also listed. These included DoD (especially 5000 series), service, and command regulations and instructions.

Most interviewees also cited standards, guidebooks, and operating instructions that applied locally or at an intermediate service organizational level (such as a major command in the Army or Air Force). The local- and intermediate-level policy documents covered a fairly wide range of activities and products, including

[13] J. Forester, "Selling you the Brooklyn Bridge and Ideology," *Theory in Society*, September 1981, p. 746; J. Forester, "The Policy Analysis—Critical Theory Affair: Wildavsky and Habermas as Bedfellow?" *Journal of Public Policy*, 1982, No. 2, p. 151; J. Habermas, *The Theory of Communicative Action*, 1984, Vol. 1, p. 308; S. Seidman, ed., *Jurgen Habermas on Society and Politics: a Reader*, Boston: Beacon Press, 1989, p. 154.

[14] MIL-STD-498 replaced both DoD-STD-2167A (for weapon systems and other mission critical applications) and DoD-STD-7935A (for automated information systems) and brought these two areas together under one standard.

documentation, coding, inspections, software quality assurance, process definition, formal reviews, project planning and tracking, configuration management, product engineering, and testing. In addition, Air Force sites described a set of directives from the office of the Deputy Assistant Secretary of the Air Force for Communications, Computers, and Support Systems covering process assessment and improvement, software reuse, and metrics. (As noted earlier, these directives have since been rescinded.)

It was clear that the most important source of policy for software maintenance was the military standards. The single most important reason was that the military standards provided a consistent framework of expectations for software developers and software maintainers—two communities that generally have limited interaction during software development. It is on the basis of what is described in the military standards that the software maintenance community "knows" what to expect in the way of software documentation. The considerable unease we found, in almost all of the interviews, regarding the demise of the military standards stems from the potential loss of this consistency of expectation. The F/A-18 program was an exception because there is almost no wall between developer and maintainer.

Policy-Related Concerns

Echoing the unease previously described, the most widely expressed concerns were related to former Secretary Perry's 29 June 1994 memorandum discouraging the use of military standards and calling for greater use of commercial standards and performance specifications. The objective of that memorandum was to orient DoD to specifying system functionality and performance rather than the process by which a system is developed. There were two concerns related to the Perry memorandum:

- There was a perception that there were no commercial process standards for software. Hence, there was nothing to replace DoD-STD-2167A and MIL-STD-498.

- There was a perception that performance specifications were not sufficient to maintain software, the design of which is constantly evolving; design and other documentation is needed.

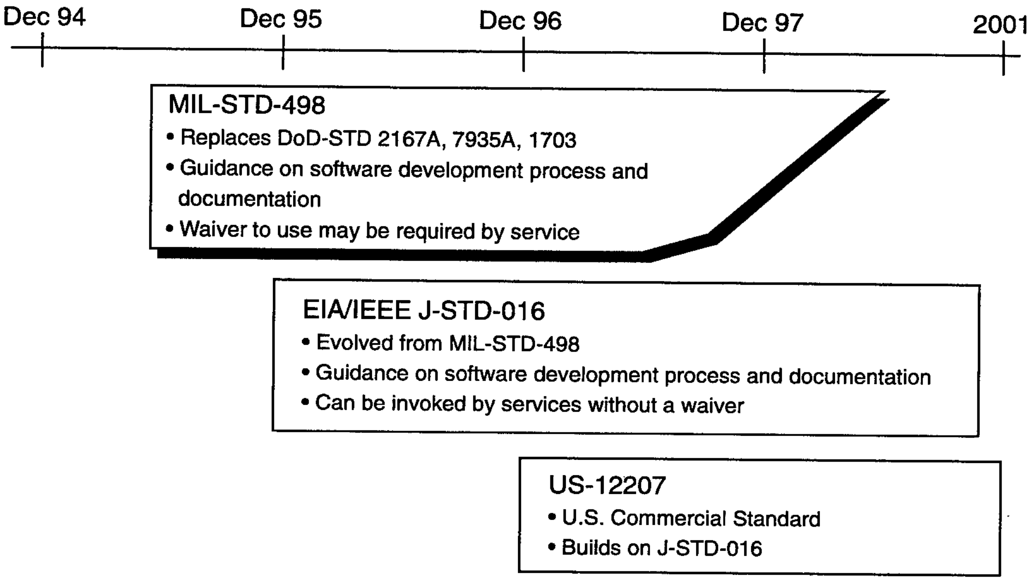
In attempting to address these concerns, the Air Force and Navy granted blanket waivers allowing the use of MIL-STD-498, initially until December 1996, then indefinitely. The Army required waivers to be granted case by case.

The legitimacy of the first concern is questionable because commercial software process standards do exist. The IEEE and Electronics Industries Association (EIA) have developed a commercial counterpart to MIL-STD-498 called J-STD-016.[15]

[15] Reed Sorensen, "MIL-STD-498, J-STD-016, and the U.S. Commercial Standard," *Cross Talk*, June 1996, pp. 13–26.

Additionally, in August 1995, ISO/International Electrotechnical Commission 12207 *Information Technology and Software Life Cycle Processes,* was released as an international standard. An adaptation for the United States is currently being developed as US 12207. The approximate time sequence of these standards is shown in Figure 3-3. We conclude that the concern over the demise of the military standards at least in part is a result of uncertainty over change, although we would not want to minimize the importance of this uncertainty. DoD-STD-2167A was very prescriptive in terms of required documentation. MIL-STD-498 effectively only suggested documentation. J-STD-016 like MIL-STD-498 also only suggests documentation. The likely scenario under J-STD-016 is that contractors would be required "to develop the software in accordance with best commercial practices." The contractors, in their proposals, would then likely propose using J-STD-016, and cite the documentation that they would produce. This will likely lead to more inconsistency (and attendant uncertainty) as to what can be expected in the way of documentation to support maintenance. This scenario mitigates for increased involvement of maintenance activities in the development of project requirements. Alternatively, the policy recommendation might be to forego organic and/or third party support of any kind (especially for embedded systems) and plan long-term OEM support.

Figure 3-3. Standards Evolution



MIL-STD-498

- Replaces DoD-STD 2167A, 7935A, 1703
- Guidance on software development process and documentation
- Waiver to use may be required by service

EIA/IEEE J-STD-016

- Evolved from MIL-STD-498
- Guidance on software development process and documentation
- Can be invoked by services without a waiver

US-12207

- U.S. Commercial Standard
- Builds on J-STD-016

The second concern appears to have solid foundation in fact. The point was made by several of those interviewed that performance specifications are not sufficient for software maintenance. In the words of one Navy software manager: "The problem with doing away with specifications is that, for us, it's not just a matter of specifying performance but also documentation if we have to maintain it organically." (Although he did not say so, clearly the same observations would apply to any software maintainer other than the OEM.)

The military process standards call out a set of data item descriptions that, at least notionally, provide such documentation. Although several interviewees acknowledged that there was often inadequate tailoring (resulting in documentation overkill), at least they knew what documentation would be available and in what format. Now they can no longer count on this. This is an understandable concern because software is typically developed by one government organization, then moved to another for maintenance.

These concerns lead to another widely expressed issue. Software maintenance organizations perceive themselves as having no real input during development, even though decisions are made that impact software maintainability. These decisions include choices of programming languages, type and extent of documentation, host and target computers, operating systems, software architectures, and buy-or-build options. We heard several examples of decisions that were made—or not made—during acquisition that have had major negative consequences for the government during the maintenance phase.

One Navy software manager told of a case in which the government had failed to specify delivery of source code (the actual human readable programming language) and received object code only (the machine readable version). This meant that the code could never by modified or enhanced by anyone except the original developer—since they owned and were the only ones with access to the source code.

The Army provided another example. The Kiowa Warrior helicopter is currently in production. Some of the onboard software was commercial off-the-shelf (COTS) and some was developed under contract to the Army. To quote one of the interviewees: "The PM is doing a study to see who owns what for the Kiowa Warrior and to determine who should provide the support. It's not clear what is COTS and owned by the contractors and what is government owned. That was never specified clearly in the contract." Once production of the Kiowa is completed in 1999, responsibility and dollars were to move from the program manager to the Army Aviation and Troop Command (ATCOM). In the interim, personnel at ATCOM could only make suggestions and perceived themselves as having limited power to initiate actions.

This "throwing the software over the wall" was a situation we heard described in most of the site visits. In the words of one Army software manager,

> Program managers have no incentive to be concerned with the long-term supportability of their system. The weapon system PMs are measured on the extent to which the system is fielded on time and within budget. The PMs need an incentive to be concerned with software supportability either because they have life-cycle responsibility or because the people who are responsible for software maintenance have a vote.

That manager went on to say

> There is nothing in the DAB [Defense Acquisition Board] approvals that addresses post-deployment software support. The PM is never challenged on the PDSS concept or costs. A PDSS gate is needed during development. The kinds of issues that should be addressed include costs, who will be responsible, and where software support will be done.

We did see a counterexample to this approach: the Navy's F/A-18 program at China Lake, CA. Unlike most of the sites we visited, the same government organization (the Weapon System Support Activity within the Naval Air Warfare Center) is involved throughout the life cycle of the aircraft. They described a joint contractor-government team in which the contractor has the lead until production is complete, then it transfers to the government. The A/B model has been fielded since 1984; hence, the government has the lead responsibility for this model. The C/D model is just completing production, and responsibility is about to transfer from the contractor to the government. The E/F model is still under development and under the lead of the contractor. The government and contractor follow a common process for all models so that people can and do move across models as needed.

The F/A-18 people that we interviewed were not concerned about the Perry memorandum. They have a tailored version of DoD-STD-2167A included as a technical memorandum in their contract and do not view themselves as being impacted by the elimination of government standards. This is a key point: there is nothing in the Perry memorandum to prevent government personnel from specifying exactly what products they want delivered, including documentation, but there has to be the knowledge and action taken up front to specify these products if they are to be delivered as part of the system. However, because most government people responsible for software maintenance indicate that they have no real input during development and little confidence that the right knowledge is in place to specify the needed documentation, their concern about the disappearance of military standards is understandable. With those standards in place, they at least knew what documentation they would be getting.

A final concern we heard is that, in some cases, policies are accompanied by insufficient implementation guidance. This concern was heard primarily from the Air Force, which is not surprising considering that it has been the most ambitious in its software policies. To quote one of the Air Force people tasked with implementing the policy related to the CMM,

> There is current policy covering software reuse, metrics, and process maturity. The policy says what but not how. There is a real need to help organizations institutionalize these things and give guidance on how they work together. The guidance must address three levels: (1) organizational, (2) project, (3) individual programmer. Currently there is too much in the way of high-level policy and not enough implementation guidance. The policy should give organizations ammunition to get the resources needed for implementation.

It was noted earlier that these policies are no longer officially in effect, but the concern over lack of implementation guidance in general is valid.

Advances or Initiatives

During the site visits, people were asked to describe any advances or initiatives within their organization that could help other software maintenance organizations. Two types of initiatives appear noteworthy in terms of demonstrated cost savings.

The first involves contract consolidation and competition. This was mentioned by both the Army's CECOM and the Air Force's SSSG (Colorado Springs). The two organizations are somewhat similar in that both rely on contractor support and both are responsible for numerous systems. Over the past few years, the two organizations have moved from individual contracts with many different contractors, each maintaining single systems, to fewer contractors, each maintaining multiple systems. These larger, "omnibus" contracts are competed, typically at 5-year intervals. Two obvious benefits were mentioned in the interviews: (1) government management overhead has been reduced, with only a few large contracts instead of many smaller ones, and (2) the competitive bidding for these larger contracts has kept contractor rates down. For example, the Defense Satellite Program at Colorado Springs consolidated seven contracts into one, which was competitively awarded to Loral. Historically, $15 million has been spent per year maintaining these seven systems. The people we interviewed expected this figure to drop to $5 million as a result of the contract consolidation.

A second type of successful initiative, cited by the F/A-18 program, entails persistent, long-term process improvements. The F/A-18 program began to improve in 1991 by getting control of the requirements process. In addition to strict control over requirements, the F/A-18 program has made more efficient use of labor, since that is their biggest cost driver. A typical operational flight program for the F/A-18 that cost $210 million in the early 1990s now costs $140 to $150 million, a 30 percent savings.

Initiatives mentioned by other sites include the use of electronic documentation rather than paper and the use of computer-aided software engineering tools to describe software architectures. No specific cost savings were cited for these.

LESSONS LEARNED

The people we interviewed were asked to reflect on any lessons learned about software maintenance. The most commonly given response was the need for effective communication with users. From the responses given, "effective communication" covers a variety of areas, including a clear understanding of requirements on both sides, thorough training of users in the operation of the system to avoid problems being reported as software defects that are actually user errors, and ensuring that users do report real problems. Different means for achieving effective communication were described, most of which involved face-to-face interaction with users. In some cases, this was achieved by collocating with users and, in other cases, by visiting user sites. The Army's ATCOM mentioned having Apache helicopter pilots attend technical interchange meetings to provide input "and they caught a lot of things we would have missed." A variety of electronic means were provided for communication with users, including hotlines, e-mail, faxes, and telephone.

The next most common lesson learned was the need for good documentation. This was also listed as a frequent problem and was the basis for the often-expressed concern that the absence of military standards would make this problem worse.

Two sites mentioned the need to document the software maintenance process and measure how much effort and dollars are being spent on each step. In this way, one could focus cost savings efforts on the parts of the process that are consuming a disproportionate amount of effort.

Two sites (both of which maintained embedded software) mentioned the value of collocating hardware and software engineering.

CHANGES DESIRED

There were no universal themes to the changes that the interviewees would like to see. However, the need for several changes did emerge often enough to be reported:

- The organizations need the ability to invest in software process improvement. This was expressed in the context of the need for training to attain CMM-level certification.
- The CMM process needs to be institutionalized. This would provide structure and high-level guidance and support for software process improvement, especially as it relates to software maintenance.
- The maintenance providers need to have a greater voice during development decisions affecting software maintenance.

EFFECTIVENESS

Although there is not yet any general sense of what distinguishes effective and ineffective software maintenance organizations, there are candidate criteria. Certain organizations we visited had a better sense of what they did, and were better able to explain it, than others. On reflecting on what seems in some sense to be consistent patterns of behavior across these potentially more effective organizations, the following characteristics stand out:

- They take commitments seriously and are able to follow through.
- They are able to articulate organizational objectives for improvement and to follow through with actions to reach those objectives.
- They participate throughout the system life cycle, not just after deployment.
- They make effective use of contractor support, competing contracts when appropriate and fostering productive long-term relationships among sole-source providers when applicable.
- They have a critical mass of people and resources along with strong, central leadership within the organization.

Of the organizations we visited, the F/A-18 program stands out as having all of these characteristics. Not only does this program embody all five, it has moved from near-disaster to a state of health within a 5-year period. According to the chief engineer of the F-A/18 program,

> The massive improvements started in 1991. They were always slipping schedules. They weren't delivering a good product. They got burned really bad on one OFP [operational flight program] and realized they no longer had an endless budget. The incentive was to stay in business. Costs were spiraling. Now we deliver a better product which is an even more important incentive.

Because the F/A-18 program has all five characteristics, it can serve as a case study of best practices.

Commitments Taken Seriously

What is striking about the F/A-18 program is the degree to which they take their commitments seriously. This is manifested in several ways.

First, there is an emphasis on requirements management. The chief engineer told us that this was the area they got under control first because requirements "creep" had been such a problem in the past. In fact, when the chief engineer first joined the program, it was unclear what requirements were being addressed in what block. They have implemented a formal process of defining and documenting requirements and of associating cost and schedule estimates to those requirements. Any changes are agreed upon in writing since "requirements creep" led to missed schedules and cost overruns. In the words of their chief engineer, "everything we do is in writing and we don't do anything different unless it's in writing."

Second, there is an emphasis on cost and schedule estimation, planning, and tracking. Detailed plans are drawn up and schedules are tracked to the day. A lag of a single day is reported to the PMA.

Finally, they rely on objective completion criteria. For example, readiness for delivery to the field is decided on the basis of defect data from testing normalized by the amount of testing.

Commitment to Process Improvement

We noted earlier that process improvement requires a commitment of time and resources. Meaningful improvements are often precipitated by a crisis. This was clearly the case for the F/A-18 program. Process improvement is an ongoing concern for this program. The block manager and chief engineer meet weekly to discuss how to improve their processes. The team leaders meet twice a week. As noted earlier, the F/A-18 program can point to real savings.

Participation Throughout the System Life Cycle

We have already discussed the concern raised in multiple interviews that the people who will be maintaining a software system typically have limited involvement during development. Decisions are made during development that impact maintenance—including the language to the host environment, the target processor, and the documentation requirements—in short, everything with which the maintenance organization is concerned. Once a system begins or ends production (depending on the service), a transfer of responsibility occurs and the system enters its maintenance phase. From the perspective of the maintenance personnel, the system is being "thrown over the wall," and they are on the other side.

The F/A-18 avoids this by involving maintenance personnel during the development of any given model. Both contractor and government personnel work on all models during all phases of the life cycle. What varies is who has the lead: the contractor has it for models before the end of production; the government has it once a model has completed production. But both organizations are involved in pre- and post-production activities to ensure a smooth transition of responsibility.

Effective Use of Contractor Support

As discussed earlier, whether contracts can be competed or not depends on the complexity of the hardware-software system being maintained. In the embedded world, we never saw a case of successful competition. We saw one program (the Navy's F-14) that had tried to compete the maintenance and repeatedly had the OEM emerge as the single bidder. Given the likelihood of this result in other cases, the best course is to foster an effective working relationship between the government and the OEM. This is what the F/A-18 program has done. (In this case, the OEM is McDonnell-Douglas.) In the words of the chief engineer,

> We're not interested in putting McDonnell-Douglas on report. We're interested in getting the product to the fleet. In order to build up trust, we don't set up competitive situations. It's important to define the roles and responsibilities for each player and ensure that people don't violate that role and engage in turf grabbing.

In the nonembedded environment, competition among contractors is possible and we saw instances in which competition was effectively used. Both CECOM and SSSG have produced significant cost savings through their initiatives to consolidate contracts into larger chunks and compete them.

Critical Mass Along with Strong, Central Leadership

The organizations that appear to be the most effective all had a critical mass of the necessary people and resources. The threshold, if there was such, appeared to be about 1,000 people total (contractors and organic) or a $100 million budget per year. This is not necessarily the rule, but there does appear to be some critical mass that is needed in order to pull together the resources needed to invest in meaningful process improvements.

Also important is strong, centralized leadership within the organization. The more effective organizations had strong leadership that could articulate priorities and future direction and had a track record of following through with action. In contrast, we saw one organization that had the required people and resources but very decentralized decision-making. The result was multiple stovepipes within the organization and a lack of cohesive direction or a set of priorities for the organization as a whole.

Appendix A
Software Maintenance Organizations Visited

The sites visited and the specific systems or types of systems being maintained at those sites are shown in the Table A-1. Information is also included about the maintenance provider for each site (contractor or organic). (The managers interviewed were always government personnel, including those cases in which the bulk of the work was done by contractors.)

Table A-1. Maintenance Information

Service	Organization	Location	Area of responsibility	Contractor or organically maintained
Air Force	Consolidated Integration Support Facility	Peterson AFB	Satellite control systems	Predominantly contractor
	Warner-Robins Air Logistics Center	Robins AFB	Electronic warfare, F-15, ATE	Mixed contractor and organic
		Scott AFB	Conducted process capability assessments of Air Force organic organizations	Not applicable (we spoke to process assessment group only)
Army	CECOM	Fort Monmouth, NJ	Communications	Predominantly contractor
	ATCOM	St. Louis, MO	Apache and Kiowa Warrior helicopters	Contractor
Navy	North Island NADEP	San Diego, CA	Helicopters/ATE	Mixed contractor and organic
	F-14 weapon system support activity	Point Mugu, CA	F-14	Mixed contractor and organic
	F-18 weapon system support activity	China Lake, CA	F/A-18	Mixed contractor and organic

Appendix B
Interview Outline

Data on DoD software maintenance practices were primarily obtained through semistructured interviews of personnel working in software maintenance. This appendix presents the interview outline.

1. Who are your software users and other customers?

2. For what system do you provide mission critical software support and what is its function?

3. How long has the system been fielded and when have major software upgrades been accomplished?

4. Describe the nature of the software support that you provide. (If appropriate, describe in terms of the IEEE categories of *corrective* [bug fixes], *adaptive* [rehosting, etc.], and *perfective* [incremental improvements]).

5. What percent of effort is in each category.

6. How many lines of executable code do you support? Please provide by system or major subsystem that relates to your support approach. What computers and software languages are used for these systems/subsystems?

7. Who performs the software support? (Please provide the number of people involved by category, organic [military and civilian] or contractor, as applicable.)

    a. What are the typical grade levels/grade structures operative in organic software maintenance activities, e.g., GS-12, WS-15? What are the occupational codes of government personnel? What is the annual cost of support in these categories?

    b. What is the basis for selecting contractor or organic support, e.g., organic or OEM/developer selected as logical choice, competition, other?

    c. What is a typical progression of software maintenance providers after initial fielding of system, e.g., OEM initial support progressing to organic support with OEM assistance?

8. What is the workload measurement basis for determining software maintenance requirement, e.g., direct labor hours per year, dollars per year, tasks per year?

9. How are organic and contract activities workloaded, e.g., work orders, task orders, job list?

10. How are these activities budgeted, i.e., level of effort, work package/task order, etc.?

11. Describe your infrastructure, i.e., facilities, tools, capabilities, etc.

    a. What types of special equipment and facilities have been established to support software maintenance requirements, e.g., support environment, networked work stations, subsystem labs, system labs, operational hardware?

    b. What are the funding sources for capital improvements, e.g., little or no funding, upgrades associated with new systems, capital investment budget, amortization generated funds, individual project justification?

    c. What automated development environments do you use, e.g., integrated CASE tool sets, such as Cadre's Teamwork or IDE's Software Through Pictures?

    d. What is the value of capital equipment and facilities involved in supporting organic activity?

12. As you perceive it, what is the operable policy affecting software maintenance, e.g., formal policy, MIL-STD-498/2167A, local operating instructions, metrics, CMM, other?

13. What is the process and communication mechanism for initiating software changes and how does the user participate?

14. How often do you field block upgrades/version releases? What is the typical cycle time?

15. What information (metrics, management indicators, etc.) do you use to monitor software support, e.g., basic status, basic 5 (effort, size, defects/quality, schedule, rework), project focused decision tool metrics?

    a. How are they used? By whom?

    b. What other sources monitor software maintenance, e.g., self-contained, peer working group, program manager/agency, user, higher headquarters? What or how do they monitor?

16. Is your organization or any of your supporting contractors CMM certified? What level of certification? Was this by formal or informal assessment?

17. Is CMM useful? In what ways? Do you have a desired level targeted? How could application of CMM be improved for your situation?

18. Describe your approach and rationale for training.

    a. What is the interface between you and the development contractor for training when new or upgraded systems are delivered?

    b. How much of your software maintenance budget is devoted to this type of training? Is this included in the figures you have already given us for the magnitude of the maintenance effort?

    c. How much of your budget is dedicated to other forms of training *not* associated with support of the delivered systems, e.g., training to maintain general proficiency in software development/maintenance (accomplished through public seminars, conferences, in-house training, etc.)? Is this included in the figures you have already given us for the magnitude of the maintenance effort?

19. What other software support activities do you work with because of the system interface, the subsystems contained, embedded software, government-furnished software contained, etc., e.g., electronic warfare, intelligence community?

20. How is software support addressed in program documents/contracts, memorandums of agreement, etc.?

    a. How do these requirements get communicated to the buying activity? How effective is the communication process?

    b. How do you interface with contractor activities providing maintenance support, e.g., oversight/insight, technical interface, procurement interface?

    c. How responsive are the acquisition organizations to the supportability concerns?

    d. What military standards or other documents are typically used in contracts to specify supportability requirements?

21. What is the extent of "module" or code reuse?

22. What are the lessons learned for software maintenance?

23. What changes would you like to see affecting software maintenance? Could OSD policy changes help the situation?

24. What advances or initiatives are you developing that could help other software support organizations? How will these innovations be communicated?

25. Are we asking the right questions? What other questions should we be asking?

26. What other organizations within the command/agency do you know of that do software maintenance, based on the definition that we have used?

Appendix C
Abbreviations

ALC	air logistics center
ATCOM	Army Aviation and Troop Command
ATE	automated test equipment
C3	command, control, communications
CASE	computer-aided software engineering
CECOM	Communications Electronics Command
CISF	Consolidated Integration Support Facility
CMM	Capability Maturity Model
CORM	Commission on Roles and Missions
COTS	commercial-off-the-shelf
DMCS	Depot Maintenance Cost System
EIA	Electronics Industries Association
GPS OCS	Global Positioning System Operational Control System
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Standards Organization
KPA	key process area
MIL-STD	military standard
NADEP	Naval Aviation Depot
O&M	operations and maintenance
OEM	original equipment manufacturer
PDSS	post-deployment software support
PM	program manager
PMA	program management activity
SCF	standard change form
SDR	system deficiency report
SEI	Software Engineering Institute
SISMA	Streamlined Integrated Software Metrics Approach

SLOC	source lines of code
SPICE	Software Process Improvement and Capability Determination
SSSG	Space Systems Support Group
STEP	Software Test and Evaluation Panel
STR	software trouble report
TPS	test program set
WR-ALC	Warner-Robins Air Logistics Center

REPORT DOCUMENTATION PAGE

Form Approved
OPM No.0704-0188

1. AGENCY USE ONLY ***(Leave Blank)***

2. REPORT DATE

Nov 97

3. REPORT TYPE AND DATES COVERED

Final

4. TITLE AND SUBTITLE

Maintenance of Department of Defense Mission Critical and Mission Support Software:
A Preliminary Characterization

5. FUNDING NUMBERS

C DASW01-95-C-0019

PE 0902198D

6. AUTHOR(S)

Elizabeth K. Bailey, Emanuel R. Baker, James A. Forbes, and Donald W. Hutcheson

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Logistics Management Institute
2000 Corporate Ridge
McLean, VA 22102-7805

8. PERFORMING ORGANIZATION REPORT NUMBER

LMI– LG518T1

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

Mr. Robert T. Mason,
Assistant Deputy Under Secretary of Defense (MPP&R)
Room 3B915
The Pentagon, Washington, DC

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

A: Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

The purposes of this study were to undertake an initial characterization of DoD mission critical software maintenance; to identify policy issues; and to outline the scope and major features of potential new or revised policy. We distinguished among three categories of mission-related software. Within a category different organizations may use similar processes; across categories they generally do not. For six specific types of software maintenance, we accounted for about 16,000 government and contract personnel equivalents (55% organic and 45% contractor) maintaining 225M lines of code at an annual cost of about $1.26B. About 40 percent of the effort is corrective and 60 percent is a combination of adaptive and incremental improvement. Pure organic maintenance is the exception for any type of software; organic maintenance of embedded software is generally found only on older models of weapon systems; where attempted, competitive contract support proved both more economical and at least as effective as either sole-source contract support or organic support. Written policy consists of MIL-STDs (e.g., 2167 and 498) and local operating instructions rather than DoD instructions or Service regulations. There is a lack of consensus over what software maintenance is also depot maintenance.

14. SUBJECT TERMS

Software maintenance, software support, depot maintenance, mission critical software

15. NUMBER OF PAGES

82

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT

Unclassified

18. SECURITY CLASSIFICATION OF THIS PAGE

Unclassified

19. SECURITY CLASSIFICATION OF ABSTRACT

Unclassified

20. LIMITATION OF ABSTRACT

UL