DTIC FILE COPY

AD-A213 470

THE CHALLENGE OF SOFTWARE MAINTENANCE COSTING

Technical Report

Dr. Roland D. Kankey

AU-AFIT-LSQ-89-1

DTIC ELECTE OCT 19 1989 S D E

DISTRIBUTION STATEMENT A
Approved for public release
Distribution Unlimited

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

89 10 19 077

THE CHALLENGE OF SOFTWARE MAINTENANCE COSTING

Technical Report

Dr. Roland D. Kankey

AU-AFIT-LSQ-89-1

The views expressed herein are those of the author and do not necessarily reflect the views of the Air University, the United States Air Force, or the Department of Defense.

Approved for release as requested
in accordance with AFR 190-1.

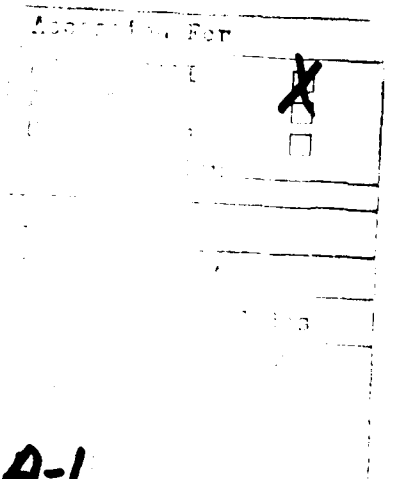
ABSTRACT

Software is already a critical element of most DOD and Air Force weapon systems and its role is increasing. From a resources point of view, its acquisition cost is growing as a percentage of the system cost, while its maintenance cost can be many times its acquisition cost. This project addressed the critical area of software cost estimation and in particular, software maintenance estimation. At this point in time software cost estimation is not well understood. A number of approaches, techniques, and models are available. This study attempted to evaluate techniques for software, and software maintenance, costing.

The sponsor of the study, Lt Col John Long of ASD/ALT, indicated that the primary area of concern was software maintenance, so the strongest focus of the literature review and the report is on software maintenance costing. A number of textbooks, technical reports, and articles were included. One of the initial problems with software maintenance is the disagreement on the concept and definition. One author considers the word "maintenance" to be inappropriate for this activity, preferring instead the words "continuing development". The rationale is that since software does not wear out, it does not need to be maintained. For many people, maintenance implies activity with lesser skill demands, yet in software "maintenance" several claim the maintainers need to have essentially the same skills as the original developers. Other phrases that are used to express this activity are software support, and post-deployment software support. The actual types of work included in each case differ slightly, as indicated in the report.

The various proposed techniques are reviewed, with prior evaluations summarized. The report ends with a select bibliography of published work in the area and a selection of definitions.

This research received significant sponsorship and support from Lt Col John Long, Aeronautical Systems Division/ALT, from January 1988 through March 1989. Reference ASD Reserve Project 88-012-DAY, Partitions for Software Costing.



THE CHALLENGE OF SOFTWARE MAINTENANCE COSTING

Roland D. Kankey

March 1989

1. BACKGROUND

In the distant past, witchcraft and spells were imagined to be required to motivate inert matter to activity. More recently, mechanical or chemical interaction was used, with widespread acceptance and understanding. The unseen tool of electricity has been in use for about a century and its mechanisms are generally understood and accepted. However, during the last four decades hardware has reverted to control of something more like witchcraft and spells. Software, once installed, cannot be measured in a normal sense. To many people, software is not well understood, let alone accepted.

What is software? One expressed view is that computer software is the pattern of electrical charges in the computer's memory and storage media. Boehm (81) defines it as the entire set of programs, procedures, and related documentation associated with a system and especially a computer system. Attempts to estimate the cost of this software over a system's life cycle have been only partially successful.

Several interesting reasons have been advanced that relate to this difficulty. Weinberg (71) indicates that computer programming can be seen as an attempt at communication between two alien species, with programming languages designed to make this communication easier for the

human species. Baker and Fisher (82) maintain that software development is seen as an art, with concurrent lack of discipline being the core of current difficulties.

While decision makers depend upon forecasts to help make better decisions, their reliance on a forecast is determined somewhat by how well it can be defended and explained. Software estimators are thus not only hampered because of the unseen product, but also because it is difficult for an analyst or manager to vigorously defend an estimate supported by questionable data, inaccurate models, and hunches (Brooks, 78). Fox (82) indicated software estimation was imprecise and personal and referred to software estimating as a black art. Woodward, as cited by Fox (82), maintained that it was almost impossible to predict the results of any development work in terms of time or money. Fox then insists this is even a stronger truth for software development, since the product is not even tangible. Software development shares this characteristic with hardware development. The time and cost to develop new things cannot be accurately forecasted.

Given that software is a critical element of most DOD and Air Force weapon systems, that its role is increasing, that its acquisition cost is growing as a percentage of the system cost, and that its maintenance cost can be many times its acquisition cost, a study of software maintenance cost is highly appropriate. This study effort is aimed at evaluating techniques for software costing particularly for the system's operating and support life cycle phase.

2. LITERATURE REVIEW: SOFTWARE MAINTENANCE

A misnomer. Software does not require maintenance in the standard sense. Yet Paul McIlvaine (82) indicated that while development might cost $75 per deliverable line of code, maintenance of that software over it's life could run $4000 per line of code. Fox (82) calls this work continued development, since it includes additions to the program and correction of latent errors. He insists that the term software maintenance understates the challenge of this work. Boehm (81) defined software maintenance as "the process of modifying existing operational software while leaving its primary functions intact".

In spite of Fox's comments, the term software maintenance seems to be most common. Brooks (78) indicated program maintenance deals with changes made after delivery. Others using the term maintenance include Boehm and Swanson (80), and Grove (82). McIlvaine (82) equates software maintenance and software support. The Joint Logistics Commanders in the Workshop 84 advanced the phrase "Post Deployment Software Support". This seems to include the traditional software maintenance as well as efforts during production/deployment that help assure a maintainable product that will support the mission. It also echoes the word support. (cited by Martin and Deasey, 87)

Software maintenance as a percentage of software life cycle cost is estimated as often 50%, or as high as 70-80% (Fox), 70% (Grove), and 50-75% (Boehm & Swanson, per Martin

and Deasy). The impact of this on a program depends upon the proportion of that program's cost that is due to software. This of course varies, but Doane (82) indicated that software costs of C^3 systems were approaching 80%. Fox (82) related that software maintenance costs could be very small or very large proportions of software life cycle costs depending upon the length of time used, the stability of the external environment, and the quality of the software when developed.

Before considering the elements and subdivisions of software maintenance, it is reasonable to review the phases of a software project and of the software life cycle. Metzger (81), in his book on managing a programming project, cites six phases:

- o Definition
- o Design
- o Programming
- o System Test
- o Acceptance
- o Installation and Operation

Notably absent from this listing is a specific phase called maintenance or support. Fox (82) only listed three phases for a program's life: Development, Use, and Continued Development. But he cited six phases for development:

- o Requirements Definition
- o Design
- o Writing the Instructions

- o Construction
- o Testing, or Verification
- o Documentation

Fox's three program life phases can be compared to the standard life cycle phases of : Research and Development, Production, Ownership, and Disposal. A first item to note is that software does not have a serious Production phase (Fox, 82). Once software is developed, it is merely copied. The costs of this copying (and perhaps printing of documentation) is relatively minor. The third traditional phase is Ownership (or Operating and Support). This phase must be compared to Fox's Use and Continued Development phases. The Ownership phase includes activities that occur after deployment of the system. Software maintenance would fall in this phase. The last phase is usually disposal. No references were found dealing with disposal costs of software, they are assumed to be minimal.

It is interesting to note that, for software to be considered equivalently to hardware, the ownership phase should include costs of both operation and support; with support roughly equivalent to maintenance. The question of whether there are differences in operating costs for different software alternatives is held open. The next section deals with the elements of software maintenance.

2.1 Elements of Software Maintenance

Software maintenance (some say modification) has been classified in several different ways. Swanson is cited for the following categories (also used by Boehm):

Corrective - correcting an error found after the software is operational

Perfective - improvements to respond to new needs or desires of the user

Adaptive - adapting to new environmental changes, such as a new operating system or new hardware

(Martin and Deasey, 87)

Corrective maintenance probably best corresponds to Class IV Modifications, which on aircraft systems are part of ownership costs. Perfective maintenance probably best corresponds to Class V Modifications, which are not considered Ownership Cost for aircraft systems. Adaptive maintenance seems to fall somewhere in between the first two in terms of appropriateness for Ownership cost. Clearly absent from the above list is a category for development work on known requirements deferred until after the software is operational. The categories given avoid the problem, but do not resolve it.

The biggest definitional difference between software and hardware maintenance is the consideration of perfective maintenance. This activity is similar to that referred to by Fox as continued development. Another aspect, cited by

Martin and Deasey (87), is that there is actually a transfer of intellectual property from the developer to the maintainer. The developer's approach and style must be available to the maintainer. As McIlvaine (82) states, the maintainer "must be capable of designing the software in the first place". The maintainer's skill level must be comparable to that of the designer.

There is some variety in the work that should be considered software maintenance. Fox (82) indicated the tasks were to add new functions, update functions, adjust for hardware updates, and to correct errors in the code. Boehm (81) discusses software update and software repair. Software repair leaves the functional specifications intact, while software update results in a changed functional specification. Boehm does limit the magnitude of changes that he feels should be considered as maintenance. Redesign and redevelopment of smaller portions (less that 50% new code) of an existing software package is considered maintenance. Design and development of a smaller interfacing software package which requires some redesign (less than 20%) of the existing software product, is considered maintenance. On the other hand Boehm does not consider data processing system operations, data entry, and modification of values in the data base to be software maintenance.

2.2 Approaches for Software Maintenance

The actual maintenance of software can be done through organic support, developer support, or independent contractors. (Martin & Deasey, 87) The approach to be used must be known early in development since each approach raises peculiar design and contract needs.

If the maintenance is to be done organically, i.e. by the user, then three items are desirable.

- o good software tools and solid documentation
- o involvement of government personnel during the system's development
- o good training program for support personnel

(Martin and Deasey, 87)

An AIA/EIA White Paper published in Concepts (Aut 82) clearly states that if the customer wishes to be able to accomplish post delivery modifications, then the software must be designed from the beginning to facilitate such customer modifiability. The White Paper also indicated the pertinent software development tools used in generating the originally delivered software need to be identified, preserved, and appropriate rights acquired. Fox (82) indicates that the function called support engineering (usually in the production phase for hardware) needs to be included in the development phas for software. Its purpose - to make the software easier to fix or change after it is in use. The White Paper also states that post-delivery modification requirements may have a significant

impact on the appropriate applications software design and the requirements for documentation and delivery of support software.

If the intent is to let the developer of the software also do the maintenance, government requirements are lessened. Failure to procure the tools, documentation, and rights discussed in the organic approach will put the government in a very weak negotiating position if any later decision is made to drop developer maintenance. McIlvaine (82) indicated that maintenance required a sustained level of effort over the operational phase of a systems life cycle. If this level can be accurately determined, the maintenance could be included in the original contract. An approach might be to include a contract option with a few discrete levels of effort and related contract prices.

If the third approach, use of independent contractors, is selected, the developer would certainly protect any procedures they considered competitive assets. The government should definitely assure that all development tools and documentation were available, and that the government had adequate rights so they could be given to a competitor.

Regardless of the desired maintenance approach, we should keep this in mind:

> There are two roles of software. The first is to make the computer hardware do the job. The second is to be amenable to change. We must control its development so it can be changed. [paraphrased from Fox (82)]

2.3 Variables that Affect Software Maintenance Costs

Numerous factors affect software maintenance costs. Referring back to the comments by Fox (82), three clear variables are the length of the operations phase, the stability of the external environment, and the quality of the original product. The effects on cost should be clear, but the causation is not one way. When these variables are known at the start of development, they will affect the product as well as the cost. Software designed for a one year application will be different than that designed for a ten year operational phase, etc.

Items discussed in Section 2.2 will also affect costs. Organic support would include different Work Breakdown Structure elements than contracted support and could well have lower costs for maintaining the same software. The trade off would be the cost of the increased effort during development. The availability of tools and documentation (and appropriate rights) will affect costs.

The inclusion of a support engineering function into development, and how well this is done, will clearly affect maintenance costs. The AIA/EIA White Paper (82) indicated that well thought out provisions for post delivery modifications were the exception.

Boehm (81) indicates that several program goals are in conflict. Development speed, program efficiency, extreme reliability, and ease of use are all seen to be in conflict with ease of maintenance. Shorter development time would

probably create the abandon-function syndrome referenced by Fox (82). As the due date approaches, more functions are deferred for later implementation. Since this later implementation is after installation, maintenance costs will increase. Efficiency is gained by better adapting the program to its environment, but Weinberg (71) cites Fisher's Fundamental Theorem

> The better adapted a system is to a particular environment, the less adaptable it is to new environments.

A highly efficient program will then be more susceptible to needing substantial maintenance as the software is expected to adapt the hardware to an ever changing environment.

Other variables that would affect cost include the type of computer system where it will run, and the specifications for the program's operation (time, accuracy). The type of software (embedded, semi-detached, or organic) will affect development and maintenance costs.

2.4 Software Cost Estimation

Metzger (81) lists twelve steps in estimating software:

1. Design the program system.
2. Estimate the total size of the program system (LOC).
3. Estimate programming manpower (LOC/manday factor).
4. Estimate (developmental) support manpower.
5. Estimate equipment costs.

6. Estimate facilities, supplies, relocation, trips, pubs, etc.
7. Add contingencies (a multiplier).
8. Consider other factors (his Fig 2.8).
9. Convert to money (yields est. of direct cost).
10. Add overhead, profit, fees.
11. Write down the assumptions.
12. Re-estimate as the project progresses.

One critique of these steps is that the assumptions should be surfaced and agreed upon much earlier in the process.

Boehm (81) lists only seven basic steps in software cost estimation:

1. Establish objectives
2. Plan for the required data and resources
3. Pin down software requirements
4. Work out as much detail as possible
5. Use several independent techniques and sources
6. Compare and iterate estimate
7. Follow up

By deleting the word software in the third step, they could be used for any estimation problem.

What are the independent techniques? Boehm mentions seven software cost estimation techniques. These are:

1. Algorithmic.
2. Expert judgement.
3. Analogy.
4. Parkinson-equate cost to available resources.

5. Price to win.

6. Top down: estimate at top, allocate down.

7. Bottom up: estimate components, sum up.

Techniques 4 and 5 might be useful for corporate estimators, but they are not realistic for us as a customer. So long as we recognize Boehm's Algorithmic technique as our standard parametric technique, his reduced list of five corresponds well to those in regular cost analysis work.

It seems the desired approach, historically, for development costs is to use an algorithmic type cost model. An OSD(C) report authored by Bailey, Frazier, and Bailey (86) indicated that model use should be mandated. Their survey reflected big advantages through use of any model. Differences among model results were found to be small by comparison to no model. They did indicate more than one model should be used whenever possible. This is consistent with recent work indicating the robustness of combinations of forecasts (Litteral, 88).

Recent discussions with Captain Joe Dean (88) dealt with the attempt to develop an Artificial Intelligence (AI) approach to software estimation. This approach would differ at a basic level from that attributed to Metzger in Section 2.4, and to most other software cost models, in that it would not be based on lines of code. This effort is consistent with the CMIP Road Map published by the Air Force Systems Command/Comptroller (85), which called for development of models not based on lines of code. Outputs

of the model are expected to include cost, lines of code, and man-months. When questioned about the use of function points rather than lines of code, Captain Dean's opinion was that function points would not be known until critical design review - which he felt was too late. The AI model will probably be designed to consider software modules, some feel for dispersion of possible costs for each module, then some overall probability statements on the output variables for the entire software project. It is not clear at this time how this model will address software maintenance.

2.5 Software Maintenance Cost Estimation

Most of the emphasis for software costing continues to be placed on development. Bunyard and Coward (82) indicated little attention was being paid to the overall life cycle cost. They cited this neglect as a contributor to further rise in maintenance costs through latent software errors (reduced test), inflexible software design, insufficient documentation, and insufficient support software.

Estimating software maintenance cost is a rather heroic venture. As Fox (82) related, software takes up the slack for the failures of other subsystems. Because it is "soft", it is easier to modify software to cover other subsystem's slips and errors. He gives an example of hardware (a satellite) that was inserted into the wrong orbit. The instruments could not work as designed. The software was

then modified to get the job done, saving years of time and the cost of a new $50 million satellite. Of course the customer complained about the $3 million overrun on the software.

One of the primary principals of estimation is to disaggregate whenever possible. Boehm (81) indicated that a large software project may contain several subprojects that operate differently and almost independently. Estimating them separately would be desirable.

A basic concept in software maintenance seems to be Annual Change Traffic (ACT) as used in the Constructive Cost Model (COCOMO). Boehm (81) defines this as the fraction of a software product's source instructions which (are expected to) undergo change during a typical year, either through addition or modification. Boehm discusses this percent as a multiplier for development man months, yielding the man months needed annually for maintenance. This of course implies that errors in maintenance estimates can derive from errors in ACT, as well as any error in the development man months estimate.

Other models, such as SLIM, use an expected pattern of software costs (the Rayleigh Curve) to estimate development and maintenance costs. The PRICE-SL model for software support uses algorithms for software support cost, but their development and nature are clouded by their proprietary nature.

Ferens (84) concluded that the quantitative accuracy of existing software support cost models could not be ascertained. His recommendation was to look into the qualitative differences between the available models, then consider their own needs and the situation before selecting a model or models.

2.6 Summary

According to Grove (82), software was once considered technical data, and was purchased in the same manner as technical drawings. Software is now recognized as a critical element of a weapon system, the element that can adjust for failure of hardware to match specifications (Fox, 82). As Boehm (82) indicates, software is an evolving field. All software cost estimation techniques are based upon previous projects. Fox (82) indicated estimates should be done by the most experienced person in the area, even suggesting outside consultants.

Thibodeau (81) recognized the difficulty of identifying analogous projects. He recommended using the simplest model structure and carefully calibrating it to represent the environment. Even then he was not optimistic as to accuracy.

The desirability of estimating software costs seems clear. However, our ability to do this is less universally supported. The first step in a program is Definition, or identification of the requirement. Doane (82) states that

we are unable to accurately state what is needed. The requirement is subject to change and redefinition on a shorter time scale than the development cycle. At best we would have an accurate estimate of inaccurate requirements. On the other hand, Metzger (81) stated that:

> Before long, reliable programs, produced within the limitations of a budget and on time, will no longer be a rarity, although it will take some time before they become an every day occurrence.

The most common metric used in software cost estimation is the expected size of the program, expressed in terms of lines of code. This has resulted in some logic problems since all programmers know they can reduce lines of code by tighter programming (additional effort). Fox (82) indicates that lines of code is prone to gamesmanship, while Boehm (82) acknowledges they do not represent the essence of the product. The AFSC "CMIP Road Map" (85) and discussions with Captain Joe Dean indicated efforts are underway to stop defaulting to this LOC metric.

Software costing, and software maintenance (continued development or support) costing in particular, are still in a very dynamic position. Significant efforts towards improvement of the software development and maintenance process have been driven by the essential and continuing role played by software in today's systems. The questionable data base for historic software maintenance, changing languages and tools, more powerful equipment, and improving software development and maintenance processes provide significant challenges to the software coster.

BIBLIOGRAPHY

AIA/EIA White Paper (82), "Suggestions for DOD Management of Computer Software", Concepts, Vol 5, No 4 (Autumn). AIA - Aerospace Industries Association, EIA - Electronic Industries Association.

Bailey, E.K., T.P. Frazier, J.W. Bailey (86), "A Descriptive Evaluation of Automated Software Cost-Estimation Models", IDA Paper P-1979; published by the OSD(C), October.

Baker, E.R. and M.J. Fisher (82), "A Software Quality Framework", Concepts, Vol 5, No 4 (Autumn 82).

Boehm, B.W., (81) Software Engineering Economics, Englewood Cliffs, NJ, Prentice-Hall.

Brooks, F.P., Jr., (78) The Mythical Man-Month, Essays on Software Engineering, Addison-Wesley.

Bunyard, J.M. and J.M. Coward (82), "Today's Risks in Software Development - Can they be Significantly Reduced?", Concepts, Vol 5, No 4 (Autumn 82).

Dean, J. (88), discussion with Captain Joe Dean at ESD on 2 December 1988.

Doane, R.B. (82), "The Evolving Nature of the C^3 Systems Acquisition Process", Concepts, Vol 5, No 4 (Autumn).

Ferens, D.V. (84), "Software Support Cost Models: Quo Vadis?", J. of Parametrics, Vol 4, No 4, Dec, pp 64-79.

Fox, J.M., (82) Software and Its Development, Englewood Cliffs, NJ, Prentice-Hall.

Graver, C.A. (85), "Air Force Systems Command Cost Research Roadmap", Prepared for AFSC/ACCE by Tecolote Research, Inc, September. Cover indicates "Cost Methods Improvement Program (CMIP) Roadmap".

Grove, H.M. (82), "Foreword to Special Edition", Concepts, Vol 5, No 4 (Autumn).

Litteral, L.A. (88), "On Combining Forecasts", ICA Newsletter, June 1988.

McIlvaine, P.J. (82), "Software Logistics: A Sleeping Giant", Concepts, Vol 5, No 4 (Autumn).

Martin, A.C. and K.M. Deasy (87), "The Effect of Software Support Needs on the Department of Defense Acquisition

Policy: Part 1", CMU/SEI-87-TR-2 and ESD-TR-87-102, January. (AD-A178 971)

Metzger, P.W., (81) Managing a Programming Project, 2nd Ed, Prentice-Hall.

Park, I.K. (85), "Software Cost Estimation Through Bayesian Inference of Software Size", Naval Postgraduate School Thesis, September (AD-A161184)

Stratland, N. (86) Controlling Software Development, J. Wiley and Sons.

Thibodeau, R. (81), "An Evaluation of Software Cost Estimating Models", RADC-TR-81-144, Final Technical Report June 1981.
(AD A104226)

Weinberg, G.M. (71), The Psychology of Computer Programming, Van Nostrand Reinhold.

Partial Listing of Software Acronyms and Definitions

Acceptance Test - Also called demonstration testing, performance test, product verification, etc. The desired outcome is written acceptance of the product by the customer. (Metzger, pg 155)

Ada - an eponym. A DOD computer language. Refers to Augusta Ada Byron, the worlds first recognized programmer. Ada did her work for Babbage's analytical engine (which was never built). Ada was the daughter of Lord Byron. (Grove, 1982, pg 24)

Applications Software - does the job, ex. payroll or navigation. (Fox, pg 46)

Big Projects - Do you need a big project? You can accomplish a great deal with a handful of <u>really good</u> people who are well managed. "If you propose a significantly smaller number of people than your competitors propose, your people obviously just don't understand the problem." "Since the customer expects big numbers, you'd better propose them." Ex. He has seen numbers tripled arbitrarily to be consistent with manpower used on a similar project. All people assigned will do something. (Metzger, pg 165/6)

Block Change Process - used for software maintenance. *Requested changes are accumulated, all are then made at* once. (Ferens, 84)

COCOMO - Constructive Cost Model (Boehm)

Continued Development - the third phase of the software life cycle. Additions to the program and fixing latent errors. (Fox, 1982, pg 7).

DEMI - Deliverable, executable machine instructions. (Boehm, pg 477-479)

Development - Effort from the beginning of product design to the end of integration and test. (Boehm, pg 59)

DSI - Delivered source instructions. DSI is expanded differently (to DEMI) on various machines. (Boehm pg 477-479)

ECR - Embedded Computer Resources. Generally involves equipment that is an integral part of a weapon or weapon system. (Grove, 1982, pg 9)

Embedded Computer Resources - see ECR and Embedded Computers. All computer equipment, programs, data, documentation, personnel and supplies integral to a defense system from the design, acquisition or operations and

support point of view [DODD 5000.29]. Excludes general purpose, commercially available A.D.P. equipment used to administer data processing [DODD 5100.40] (Grove, 1982, pg 9)

Embedded Computers - no universally accepted definition. Often described as "incorporated as an integral part of, dedicated to, or required for direct support of, or for the upgrading or modification of major or less-than major systems". Similar to mission-critical computer resources. Becker (82), pg 38.

Environment - Source of influencing forces that are external to the product being developed. (Thibodeau, pg 1-11)

Environmental Factors - experience, tools, facilities. These may appear explicitly in cost estimating procedures or, more often, they may influence the applicability of a given model to a given development environment in some unknown way. A major consideration in evaluating models for AF use is measuring the ability of the model to define the environmental parameters. This is because the AF must always make its estimates at arms length. It must know how the cost of software is influenced by how it is developed and who develops it. (Thibodeau, pg 1-2) There is very little objective basis for distinguishing among projects to determine whether they (prior projects and data) are truly applicable to the proposed environment. This capability is essential if the AF is to properly evaluate software development and maintenance proposals from different organizations. (Thibodeau, pg 1-3)

FAST - Freiman Analysis of Systems Technique

Flow Charts - The detailed blow-by-blow flow chart, however, is an obsolete nuisance, suitable only for initiating beginners into algorithmic thinking. (Brooks, pg 168)

ISA - Instruction Set Architecture

Instruction Set Architecture - The rules and procedures by which the hardware executes instructions or computer programs. Or the structure of a computer that the programmer needs to know to write time-independent machine language that will run correctly on the computer. (Becker, 1982, pg 39) The attributes of a computer as seen by a machine language programmer. (Grove, 1982, pg 30)

Libraries - A set of programs. At least three levels of control: working copies have no restrictions on their use or modification, system integration sublibrary included programs that were ready to integrate (changes only with permission), while the current version library was touched only to fix crippling bugs. (Brooks, pg 133)

Life Cycle Phases - Thibodeau compares hardware life cycle phases with software life cycle phases on page 3-10.

Life Cycle Cost - Technological advances in the computer field have ... shifted the systems life-cycle cost from hardware to software. Hardware life-cycle cost now represents only a small percentage of the total system cost. (H. Mark Grove in the foreword to Concepts, Vol 5, #4, Au 82, pg 7)

Life Cycle Costing - an issue that has yet to be resolved. The models currently used are poor, and there is very little correlation from one model to another, indicating effort is needed in the technology base. (Grove, Concepts, Au 82, pg 25)

Lines of Code - No common definition. May or may not include comment lines. May or may not include specification statements. May or may not include copied lines from prior files. (Thibodeau, pg 4-4)

Man Month - COCOMO uses 152 hours of working time per man month, or nineteen man-days per man-month. (Boehm, pg 59)

Multiprocessing - The use of more than one CPU, attached to a common memory or memories, with one operating system in control. (Fox, pg 279)

Multiprogramming - A systems software technique that manages the computer in a way to switch from one program to another without loading or unloading either program. (Fox, pg 278)

Obsolete Equipment - Equipment that no longer meets the requirements and equipment that can no longer be logistically supported. (Becker, Concepts, Au 82, pg 39)

SLIM - Software Life-Cycle Management. A model marketed by Quantitative Software Management, Inc. Estimates the development cost (which is not defined the same way in all models) for the entire system, then extends the scope to include the requirements and specification phases. The design and coding, integration and testing, and installation phases are fixed portions of the development phase. In one operating mode, SLIM is a top level estimate constructed from individual subsystem size estimates. (Thibodeau, pg 4-12)

Software - The entire set of programs, procedures, and related documentation associated with a system and especially a computer system. (Boehm, pg 16) No longer merely a part of a majo. system, but a system that performs the integration functions for the systems, whether they are avionics, or missiles, or command and control functions. (Grove, Concepts, Au 82, pg 12)

Software Development - A project, not a job shop. (Metzger, pg 91)

Software Efficiency - ex. A program spent over 10% of a $6 million computer's time for two years calculating A = B*SQRT(2). Programmer forgot to change SQRT(2) to 1.4140678. (Boehm, pg 721)

Software Life Cycle Phases - Three phases: development, use, continued development. (Fox, 1982, pg 7)

Software Maintenance - The process of modifying existing operational software while leaving its primary functions intact. Falls in the two classes of 1) Software Update which results in a changed functional specification, and 2) Software Repair which leaves functional specifications intact. (Boehm, pg 54) Approximately 70% of the (software?) life cycle cost for most weapon systems. (Grove, 1982, pg 7) The modification of program code to correct an existing error, adapt the program to accommodate an environmental change, or perfect the program by improving its capabilities. (Smith, class report, 1983, pg 4) Smith cites Munson for: 70% adapt/perfect, only 30% due to errors.

Software Maintenance Cost - It is estimated that software maintenance cost is approximately 70% of the life cycle cost for most weapon systems. (Grove, Concepts, Au 82, pg 7)

Software Repair - does not change functional specifications. Falls in three classes: 1) Corrective Maintenance of processing, performance, or implementation failures. 2) Adaptive Maintenance to changes in the processing or data environment. 3) Perfective Maintenance for enhancing performance or maintainability. (Boehm, pg 55)

Structured Programming - the design, writing, and testing of a program in a prescribed pattern of organization; rigor and structure in programming; order, clarity, readability. (Metzger, pg 87)

Structured Walk Through - An organized review of a project member's work by other project members. Developer gives a tutorial, then walks the reviewers through, step by step. Four to six participants, one of them is always a moderator. Up to two hours in a session. Purpose is to find any errors. (Metzger, pg 95)

Success Schedules - Those that are based on everything coming in on time. (Fox, pg 251)

Support Software - programs that help programmers and management create software that runs at use time, ex. compilers and assembiers. (Fox, pg 46)

System - a structured combination of interacting parts satisfying an objective. (Metzger, pg 23)

System Software - manages the resources of the computer such as the disks, main memory, CPU), ex. operating system. Fox claims that systems programs are the most difficult to develop and the toughest to visualize and understand. More complex by far than applications programs, and more complex than support programs. Fox claims systems software is nine times more difficult to develop than applications software due to logical intensity. Fox cites Brooks on this.

Throughput - a measure of the total system: the CPU, the memory, the tape drives, the software, and the operators. (Fox, pg 30)

Types of Software - Embedded, semi-detached, and organic. (Boehm)

Users - The seasoned systems manager knows that to design a system to fit the idiosyncrasies of one or a group of powerful users is to doom the system to a short life. Systems that are this personalized never survive the departure of the user they were designed for. This problem is encountered far more often in <u>military systems</u> than anywhere else. (Fox, pg 107)

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AU-AFIT-LSQ-89-1	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Department of Quantitative Management 6b. OFFICE SYMBOL (If applicable) AFIT/LSQ	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology School of Systems and Logistics Wright-Patterson AFB OH 45433-6583	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
The Challenge of Software Maintenance Costing

12. PERSONAL AUTHOR(S)
Dr. Roland D. Kankey

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM _____ TO _____	1989 March	27

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Software Costing, Software Maintenance Costing
12	05		
05	03		

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

Software is already a critical element of most DOD and Air Force weapon systems and its role is increasing. From a resources point of view, its acquisition cost is growing as a percentage of the system cost, while its maintenance cost can be many times its acquisition cost. This project addressed the critical area of software cost estimation and in particular, software maintenance estimation. At this point in time software cost estimation is not well understood. A number of approaches, techniques, and models are available. This study attempted to evaluate techniques for software, and software maintenance, costing.

The sponsor of the study, Lt Col John Long of ASD/ALT, indicated that the primary area of concern was software maintenance, so the strongest focus of the literature review and the report is on software maintenance costing. A

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
☒ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL DR. ROLAND D. KANKEY	22b. TELEPHONE (Include Area Code) 513-255-8410	22c. OFFICE SYMBOL AFIT/LSQ

DD Form 1473, JUN 86 *Previous editions are obsolete.*

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

19. Abstract (Cont'd)

number of textbooks, technical reports, and articles were included. One of the initial problems with software maintenance is the disagreement on the concept and definition. One author considers the word "maintenance" to be inappropriate for this activity, preferring instead the words "continuing development". The rationale is that since software does not wear out, it does not need to be maintained. For many people, maintenance implies activity with lesser skill demands, yet in software "maintenance" several claim the maintainers need to have essentially the same skills as the original developers. Other phrases that are used to express this activity are software support, and post-deployment software support. The actual types of work included in each case differ slightly, as indicated in the report.

The various proposed techniques are reviewed, with prior evaluations summarized. The report ends with a select bibliography of published work in the area and a selection of definitions.