Corporate Level Software Management

JOHN D. COOPER

Abstract—Software management is considered from the corporate headquarters viewpoint. This perspective encompasses all facets of management, but specifically dealt with are those that are brought to bear on software management obstacles and ways to cope with them are presented. Standardization is presented as the most effective management device available at the corporate level for enhancing the overall software posture. Corporate management actions available for favorably influencing the quality of software over its life cycle and research initiatives are described.

Index Terms—Interface specifications, life cycle costs, life cycle management plan, maintenance, management disciplines, software contracts, software management, standardization.

I. Introduction

THE U.S. Navy is a 30 billion dollar a year organization with corporate software issues very similar to those of large corporations such as General Motors, EXXON, and the Ford Motor Company. This paper will present a look at software management from the corporate headquarters staff point of view. This perspective includes overall software costs, satisfaction of corporate users of software, performance of software to achieve goals of the corporation, advanced planning for software requirements in the future, and anticipating life cycle costs of existing software programs. Software management at this level requires some techniques to achieve control which are not applied directly to the software development and maintenance process. Concepts such as standards and procedures, fiscal control, development check points, and product assurance are used by corporate management to monitor and insure that development projects are proceeding on schedule and that operational systems are performing satisfactorily.

First, some of the obstacles and pitfalls at the corporate level are presented. This is followed by a section on standardization, a discipline which is very important because of its great potential for reducing total system costs. Next discussed, arranged in order by the software life cycle phases, are various policies and procedures that are available for enhancing the overall corporate software posture. This is followed by a brief section which presents a few management control mechanisms. The paper concludes with some research and development opportunities in the area of software management.

A synopsis of corporate level software management could be stated as follows. The current state of the art is driven by the fact that digital technology is evolving faster than is our ability to manage it. However, corporate headquarters is in an especially strong position to influence, in a constructive way, the future evolution of software management.

Manuscript received June 15, 1977; revised November 15, 1977, February, 27, 1978, and April 5, 1978.

The author is with the United States Navy, Headquarters Naval Material Command, Washington, DC 20360.

II. Management Obstacles and Pitfalls

There are various obstacles and pitfalls confronting the corporate level software manager. The more significant ones to be discussed in this section are shown in Tables I and II. A glance at the lists will reveal that they are, for the most part, the growing pains of an adolescent technology. If they were technical obstacles, they could be confronted head-on. However, they are primarily educational and/or psychological in nature making them very difficult to resolve.

Probably the greatest single obstacle to corporate software managers is a lack of computer related experience on the part of corporate decision makers. When these executives received their education and served their apprenticeships in projects, computers had not yet emerged as a significant system development factor. As a consequence, they often lack an appreciation for the unique complexities of software development. Thus, when trying to establish a corporate policy, to initiate a software oriented project, or to obtain commitment to a corporate standard, the software manager is often frustrated by the old traditional hardware rationale being applied to the corporate decision making process.

A closely related obstacle is a general lack of available computer oriented expertise for management within the projects that contain computers. This is a function of the explosion in the application of digital technology within system developments which is occurring at a rate faster than the ability to grow, staff, and manage the attendant software developments and maintenance. What this means at the corporate level is that the demand for computer knowledgeable personnel exceeds the supply of this type of expertise within the labor force. To compensate for this deficiency, it is therefore necessary to provide the projects with all of the crutches, tools, and help that is possible.

Another obstacle to software management from a corporate viewpoint is the hardware orientation of the industry. Hardware came first and thus is deeply entrenched. People are familiar with it. Software is relatively new, intangible, and has evolved with an aura of the mystic. It is human nature to concentrate on the things that are familiar and to shy away from the unknown or foreign. As a result, projects are usually staffed with a dominance of hardware engineers; project managers try to treat software just like hardware (i.e., "MIL-STD-XYZ has been effective in hardware for years—I want it applied to my soft-

0098-5589/78/0700-0319$00.75

TABLE I
PSYCHOLOGICAL OBSTACLES

1) Lack of computer related experience by senior executives.
2) Lack of computer related expertise on project staffs.
3) Preoccupation with hardware by managers.
4) Preoccupation with development by project managers.
5) Inadequate student preparation by academia.

TABLE II
PROCEDURAL OBSTACLES AND PITFALLS

1) Potential for projects to become overly complex.
2) Use of assembly level languages in projects.
3) Buying-on a contract by the vendor.
4) Lack of adequate risk, cost, and reliability estimating techniques.
5) Lack of uniform software practices.

ware, too"). Quality Assurance, Reliability and Maintainability, and Configuration Management personnel try to treat software as hardware technical data. The impact of this orientation is that the overall corporate software posture is impaired.

A common pitfall is that project managers tend to be development oriented. Their most pressing responsibility is the development of their system within budget and on schedule. Consequently, they optimize the development process, often at the expense of overall life cycle cost considerations. What typically happens when costs start to overrun is that some of the documentation is cut to get back within the budget. This is followed closely by curtailing the testing and training. From the corporate global viewpoint, this practice is a clear case of false economy by being penny wise and pound foolish.

Academia could be of greater help to corporate level software management problems than it currently is. Computer science students simply are not prepared to face the real world of non-academic problems. Help is needed in the following:

1) assisting the student in making the transition from the situation where a computer program is developed by one person (himself) to a project where a single large computer program (or an interrelated set of programs) is developed by many different people;

2) making the student fully appreciative and knowledgeable in the several disciplines of quality assurance, configuration management, integration, testing, and documentation; and

3) making the student "end product" oriented instead of academic/laboratory oriented.

In contrast to mathematics which includes both theoretical and applied branches, computer science education should be faulted for not adequately developing the applied or engineering branch of the field. The computer science perspective in academia is contrasted by the software engineering perspective in industry. A merge is needed in both academia and industry. For the time being, however, corporate resources are required to compensate by providing the necessary education and training.

Moving into the area of methodological or procedural obstacles and pitfalls, the escalation of project complexity is a pitfall that is of concern at a corporate level. A great deal of discipline has to be exercised over (or better yet, by) the project manager, especially in the beginning, to restrain the project from becoming overly complex. There is a strong desire by the project manager, his staff and his contractor to use the latest, greatest, so-called "state of the art" to produce an elegant system. To unnecessarily incur the expense and the risk without sufficient cause is not good business judgment.

Project managers also are too often reluctant to say no to a user request for a change or a new feature. Instead, they accept the task of attempting too many new developments simultaneously. As one new facet of the development experiences growing pains, it has impact on other parts of the development; then some other facet of the development slips, again impacting the rest of the development, and so on, making the total system development a long, laborious, and complex task. In order to optimize their probability of success, projects should be kept as simple as possible and still be able to satisfy their fundamental requirements.

An ever-present obstacle is the assembly language syndrome. Some of the main reasons for utilizing Higher Order Languages are to improve software reliability and the productivity of programmers [1]. Yet for a variety of reasons (almost all of which are invalid), assembly level languages are still being widely used to develop software. From a corporate viewpoint, the use of assembly language is one of the greatest sins that can be committed. Its impact on total life cycle costs cannot be underestimated. It is curious that coercion by corporate management must still be resorted to today in order to prevent the unnecessary use of assembly level languages.

A very tempting but deadly pitfall is the contract "buy-in." In large organizations where there is a lot of potential business involved, contractors will sometimes underbid a competitive procurement either to get a foot in the door or to gamble on growth or project escalation. Having "bought in" on the contract (maybe inadvertently) the vendor must then minimize costs. Obviously, in these situations, just as everywhere else, you get what you pay for. Since the contractor is counting on growth to make his money, you can be sure he will do everything he can to get the contract escalated. It is very tempting to accept a buy-in, especially if the project is underfunded to start with; however, the project ends up paying for it later when funds to meet cost escalations are not included in the budget. This has an especially adverse impact on corporate planning and budgeting.

The lack of adequate risk, cost, and reliability estimating techniques and procedures are well-known obstacles not only to corporate software management but to the industry at large. At the headquarters level, these deficiencies impact product planning, competitive analysis, budgeting contracting, scheduling, and maintenance prediction. A result is the introduction of unknowns and imprecision into all major corporate functions. This is a constant source of frustration to management personnel who are accustomed to, in some cases, operating with very close tolerances. The inability to accurately analyze these software parameters is a large contributor to a poor image software possesses among corporate level managers.

Finally, corporate software management is also frustrated by

a lack of uniform software practices. When projects are managed utilizing different techniques, status reports require different interpretations, and resource requirements occur on diverse schedules. Different design methodologies use different sets of milestones. Dissimilar controls are necessary for dissimilar program building and integration schemes. Varying quality measures, reliability prediction techniques, and cost estimation procedures among systems preclude comparison. In short, it is very difficult for headquarters to stay adequately informed on the state of software within the corporation.

III. Standardization

Up to this point in the paper, only the difficulties associated with corporate level software management have been presented. While a great many of these difficulties must be endured for a time to come, there are many positive things that can be done today to improve the management of software. Also from its vantage point, corporate headquarters is in an especially strong position to influence, in a constructive way, the future evolution of software management. The remainder of this paper will discuss many of these constructive activities.

At the corporate level, standardization is probably the most powerful management device. Almost all software management disciplines, tools, and techniques are dependent upon some form of standardization. Although standards are not always optimal for every application, they are extremely cost-effective when the entire corporation is considered and they are usually well worth any local sacrifices in efficiency.

The logistical and training benefits due to standardization are widely understood. Likewise, software transferability or reusability is obviously dependent upon standardization. Integration and interfacing of separate programs requires standarization. Standardization also broadens the labor pool, facilitates cross utilization of personnel, enhances management visibility, and makes procurements simpler and less risky. In short, standardization has a very beneficial effect on corporate resources.

Standards should be viable and not allowed to stagnate. The evolution of technology is a continuous process; thus, it is necessary for standards to evolve in a series of parallel but discrete steps. Some of the high pay-off areas for standardization are as follows.

1) Hardware Devices: These include mainframes and peripherals. Transferability of software, cross utilization or sharing of equipment, easier maintenance, simpler logistics, and reduced training are enhanced.

2) Interfaces: These include both hardware and software. Standard interfaces are absolutely necessary for integrating both hardware and software. Standard interfaces are absolutely necessary for integrating both systems and programs. Transferability of software and cross utilization of equipment are enhanced.

3) Programming Languages: The benefits of standardizing on programming languages are widely understood. Its beneficial effect on compiler maintenance, the labor force, documentation, and miscellaneous administrative matters, although lesser known, can amount to considerable savings to the organization.

4) Computer Program Documentation: Documentation is often maligned, but correct and complete documentation is still cost-effective, especially from the corporate vantage point. Transferability of programmers, computer program maintenance, and configuration management are greatly facilitated. It also has a very favorable effect on the users of the documents when there is only one form to learn and use. Automatic documentation systems can be installed.

5) Support Software: This includes test tools, debug aids, librarians, etc. Portability is again an issue, but the unnecessary reinvention of any wheels can also be prevented.

6) Programming Standards and Conventions: When a single program or system is being programmed by more than one person, it is necessary that all adhere to the same set of programming standards and conventions. Not only is it necessary for interfacing with one another's code, but uniformity enhances readability and thus maintainability of the program. The same set of programming standards and conventions across all programs and systems contributes to overall corporate software transferability and economy.

Standardization is viewed by many as limiting or restricting freedom and stifling initiative. This is not true when standards are developed and applied intelligently. Most of the barriers to effective standardization are generally psychological, as opposed to technical, in nature. Some program managers feel that standards are an infringement on their prerogatives and deprive them of the latitude that they need to do their job effectively. Even though allegiance to standardization is sworn to, the following are some of the more common excuses given for not complying with the standards.

1) Because our problem was unique, it was necessary to deviate.

2) It was not feasible to comply with the standard at that time.

3) We will start out using this more expedient nonstandard and will convert to the standard at a later, more opportune time.

4) An excuse that deserves a little elaboration usually is heard in one of the following forms: this is a one-shot program and it will be thrown away after a very short period of time; *or* this program will be like a black box, none of its functions will ever change; therefore, it will never need to be modified. This rationale is pure mythology. These kinds of programs have a long history of hanging around for years and requiring significant modification and maintenance (coupled, of course, with associated expenses). Short circuiting the stardardization requirement always winds up being an expensive expedient.

Standards cannot be expected to be complied with voluntarily. Enforcement mechanisms are required to cope with the above kinds of excuses and many others as well. The enforcement job is very unpopular, but necessary because standards do save money.

IV. Management Actions

One of the basic managerial objectives is to reduce or minimize costs. Thus, the thrust of most corporate software management initiatives should, in one way or another, be related to reducing the total life cycle costs of a system. The emphasis is on *total* life cycle. It has been estimated that 75 percent of the cost of a system at General Motors is involved

in the operational or maintenance phase of its life cycle [2]. Yet maintainability considerations for this phase are often compromised by project personnel in order to facilitate efficiencies during the development phase. Corporate managers should guard against these myopic tradeoffs and more importantly, they should be prepared to "front-end load" a system development anticipating the return on the investment to occur during the system's maintenance phase.

Since software is critical to the functioning of any automated system, it should receive major attention by corporate management. There are numerous actions available at the corporate level for favorably influencing software over its life cycle. As used herein, the software life cycle is divided into the following phases: specification–where it is decided what the corporation will be gaining in return for the resources expended; design–where the software's usefulness to the organization is determined; programming–the period of software construction; integration and test–where it is discovered what the corporation actually received; and operation and maintenance–the period where return on investment is realized. Following through these phases of the life cycle, the next few paragraphs will discuss some of these management actions. There are two points to be kept in mind: 1) these are corporate level actions, and not an all inclusive list of detailed actions that should be taken; and 2) most of them occur during the early phases of the life cycle. Much of what is done early in the software life cycle has impact throughout the remainder of the life cycle. This is why it is so difficult to step in when a system is having difficulty and put it back on track.

In spite of the software orientation of this paper, it should be pointed out early that the customary production management disciplines of Quality Assurance, Configuration Management, and Reliability and Maintainability are fundamentally the same for software as for hardware and should be applied in software developments. However, some of the implementation techniques within these disciplines are necessarily different. For example, Quality Control disciplines should be applied to both hardware and software. In hardware, there are micrometers, calipers, ohmmeters, gauges, and so forth to check certain measurements to assure they are within tolerances. On the other hand, these tools will not work on software and there are no adequate counterparts. Thus a very different set of quality assurance techniques have to be used for software. In the configuration management domain, identification and auditing procedures are vastly different. The physical configuration audit for software is not yet adequately defined. In Reliability and Maintainability, mean time between failure (MTBF) and mean time to repair (MTTR) are widely used hardware parameters, but in software their values cannot be ascertained. Among project personnel, software-oriented people tend to feel that software is so different that all of these disciplines require complete modification in order to be applicable. On the other hand, the hardware traditionalists feel that software management is no different from hardware management, and therefore, no modification of these disciplines is required. Both groups are wrong, of course. Corporate management should recognize that the truth lies somewhere between these extremes and they should insist upon the appropriate application of these effective disciplines.

Specification Phase

Before the acquisition is initiated, it is important that it be thoroughly planned. It should be a corporate policy that all project managers be required to develop (and then adhere to) life cycle mangement plans for their systems. These plans should contain a lot more than just a layout of finances and schedules. They should include software development plans with a detailed itemization of the work to be performed and the associated milestones, configuration management procedures that institute control of change and integration at an earlier time than is now customary, and they should map out all verification and validation or quality assurance procedures starting on day one and lasting throughout the total life cycle. The life cycle management plans should also contain resources/facilities utilization schedules and considerations. They should provide for test plans that ensure that testing is timely, complete, systematic, and repeatable, not leaving this regimentation to chance. Further, they should plan in detail for transitioning the software from development to maintenance, especially if responsibility changes from the developer to a different maintenance organization. Finally, all aspects of the system's operational or maintenance phase must be laid out. Throughout the entirety of the life cycle management plan, it is critical to ensure that all schedules are realistic.

With an approved life cycle management plan in hand, it is then time for the project manager to draft his solicitation for a contract proposal. (Even if the development is to be a totally in-house effort, it is very profitable from the corporate point of view to formalize the developer-buyer relationship by requiring a written "in-house" contract-like document between the two groups.) Solicitations and their responses often get turned into contracts, so the software coverage must be complete; subsequent contract changes are very expensive. A common pitfall is to fail to include in the contract as a deliverable, some piece of support software, only to discover later that it is necessary for operations or maintenance. Such an acquisition subsequent to the original contract is usually very expensive. An especially valuable corporate management policy is to require that the team that drafts the solicitation include a person from the organization that will ultimately be responsible for software maintenance. That individual will have a very selfish motive for ensuring that the software is well covered.

Another good policy to be applied when drafting the solicitation is to specify (or plan to furnish to the contractor) specific items of existing hardware and software to the maximum extent possible. An effective corporate standardization program makes it much easier to draft the solicitations and specifications. Additional savings can be realized by specifying the use of off-the-shelf hardware and software technology for system development. There are times when research and development (R&D) to advance the state of the art is appropriate as in the case of achieving a competitive advantage. But as a general rule, it is better to forego the R&D risk and expense and capitalize on such investments of others.

In recognition of life cycle maintenance, the acquisition contract should be required to contain a provision for the capability to expand critical system resources at a time subsequent to delivery of the system. In addition to needing such availability for future corrections, enhancements, and modifications, this requirement has the favorable side effect of reducing cost and risk. It is well known that as a program approaches the physical limits of the computer, shoehorning in the final functions causes the cost and complexity of the program to increase dramatically [3]. Reserved resources and expandability provide a buffer against this phenomenon. Another beneficial side effect of the specification and costing of expansion requirements is that all levels of management attention are focused on these resources. Program development visibility is thus enhanced.

Higher levels of management can contribute to a project's probability of success in the area of staffing. When initially staffing, it is important that a high percentage of the personnel have strong experience in the applications area being addressed. Another important function that corporate level management can perform for the project is to strive to ensure stability of project personnel. This includes both in-house and contractor personnel. Continuity is an important consideration. It is very expensive and time-consuming to train a new person to the same point of expertise as the person replaced. This is not news to technical project managers in any profession, but it bears repeating for the benefit of corporate managers.

A management action that can have very beneficial effects on the development process is the use of independent, external project quality audits or reviews. There is a natural resistance to audits when they are first initiated, but this is quickly overcome as the project manager and his staff begin to realize the resulting cost savings and development efficiencies. These audits or reviews should be initiated early in the development cycle while there is still ample time and opportunity remaining to make constructive changes. Late in the development process, the majority of the project parameters have become fixed, and corrective changes are difficult and expensive to make. The objective of audits is to improve the quality of the system by insuring conformance to prescribed policy standards and disciplines. When reviewing project requirements, specifications, plans, and procedures, various inconsistancies and gaps, as well as errors, are frequently uncovered. The net result is a more efficient system development.

Design Phase

Heavy emphasis by both corporate managers and project personnel should be placed on the design phase of a development project. The old adage of an ounce of prevention being worth a pound of cure is especially applicable to a software development. A Verification and Validation or Quality Assurance plan should already be in execution by the project manager and his contractor. The group performing this function should not work for the project manager in order to be protected from any potential conflicts of interest or coercion. Coercion is particularly apt to occur when a project is in a funding or scheduling crisis.

The following are several project initiatives that should be fostered by corporate management. Involvement of the eventual maintainer in the monitoring of the project is to be encouraged. His expertise and selfish interest should once again be utilized. A disciplined program development methodology with incremental integration and testing should be mandatory. The system design and architecture should be validated by modeling, simulation, or even manually prior to being implemented. Not all software architectures are successful. There are a few that just cannot be made to work, and there are many that will require a great deal of unnecessary change (and poor design as the cause may not be readily apparent). This could yield an especially dangerous situation because faulty designs and architectures normally do not show up until system integration or installation when there are neither time nor resources remaining for repair.

A vital corporate policy during the design phase is the requirement for documentation of interface specifications. This includes *all* software to software, and system to system interfaces. The key to success of these interface documents is that they be *jointly* developed by people from both sides of the interfaces and not be done unilaterally. They must be frozen before the system can be completed. As soon as they are completed, they must be placed under strict configuration management since a fluid interface could have serious repercussions throughout the system over a long period of time.

Finally, programming in parallel with design should not be permitted. There is always a temptation to get started too early with the coding. Coding gives some tangible way for the project to demonstrate progress, but it must be resisted. Not only does it yield a lot of wasted time and resources as the design changes, but it also builds in a subconscious reluctance to make necessary design changes. It should be required that any design to be programmed should be first completed (excepting "STUBS") and formally reviewed prior to commencement of coding. Headquarters should take this procedure into consideration when reviewing the project's progress.

Programming Phase

Prior to the commencement of programming, a corporate-wide programming standards and conventions manual should be placed in the hands of all programmers. In addition to providing discipline within the development process, this document provides most of the necessary ingredients for software transferability. The ground rules for achieving program modularity are also spelled out. Included, too, are the protocol rules to be followed for interfacing modules, programs, etc. This manual is every programmer's bible.

The use of Structured Programming or a similar program development discipline should be mandated from corporate headquarters [4]. The idea of code walk-throughs, *a la* the egoless programming concept, has proven to be extremely valuable [5]. Not only have walk-throughs helped to discover programming errors and interfacing problems early when they are easiest to correct, they have helped in the removal of the veil of secrecy from around the programmer. From the coporate viewpoint, the wizard or prima donna

is not to be tolerated on production-engineered software projects in the organization. Not only is this type of programmer a personnel management problem, but generally his secret code violates all standards and conventions and can only be maintained by him.

Performance monitoring by both hardware and software techniques should be a component of the corporate tool bag. Its usage should be instituted during the programming phase and then continued throughout the remainder of the system's life cycle. Although these tools have been in existence for several years, they are still not widely used. One typical reason is that project personnel cannot convince corporate management to expend the necessary resources to obtain them. Performance monitoring is invaluable for identifying timing problems, bottlenecks, and system loading. Once installed, these tools can benefit all of the systems in the corporate inventory.

Integration and Test Phase

It is important that the testing be conducted by someone other than the developers. From the headquarters level viewpoint there are three good choices—a separate test group, the Quality Assurance group (if a test group does not exist), or the software maintenance organization. Testing must be objective, for developers can be paternal and biased. The testing must be against the original requirements, not the program "as built." Real world testing must be insisted upon. At some point, it is desirable to involve the ultimate operators and users in order that they may subject the software to the type of treatment that it will receive after acceptance. Remember, it is the user who must be satisfied in the long run.

During the integration and test phase, the need to have total configuration control cannot be overemphasized, for testing during this period will uncover new software errors. The daily configuration control problem becomes one of keeping an accurate status of all errors, control of all changes to programs and documentation (including changes to the changes), and progress of program integration. For large systems, this process must be automated or it will be excessively cumbersome. The configuration control process is a critical function, and as such, deserves considerable corporate management attention.

Operations and Maintenance Phase

Since most large organizations have more than one automated system in their inventory, the overall corporate maintenance strategy plays a crucial part in many of the decisions of the functional system acquisition. Most critical is whether life cycle maintenance is to be performed "in-house" or external to the organization. When maintenance is to be performed by the original developer, it is important when making acquisition decisions not to allow yourself to become addicted to his support. No matter how large an acquisition, your system will not always be his number one priority. At some point, whether it be one year after delivery or five, he will be unable to keep his best people on your job. He will have more immediate crises to resolve, and he will need systems where he can assign new employees for training. Furthermore, normal personnel attrition is to be anticipated. Someday, you may likely become dissatisfied with the contractor's performance and would like to be in a position to replace him.

The motivation for in-house maintenance is to reduce costs. A contractor must make a profit to stay in business. He is an expert in his business and may, in fact, be able to do the maintenance more economically than you. However, the true economics must tell the story, thus, the analysis leading to a maintenance decision must be done honestly and on a full cost basis. There are other factors that this analysis should take into account such as delays in contracting, manpower ceilings, and peak workload periods. There is a nuisance factor to consider as well; the organization might have to deal with a different contractor for each system in the inventory. At best, this is a very complex issue and should be decided very carefully.

Deserving of considerable corporate interest is the transition of software from development into operations and maintenance. This evolution must be meticulously planned well in advance of system turnover. The maintainer must have in place all of the facilities and tools needed to operate, test, and maintain the new programs. This, in some instances, may require lead times of up to 2 years. The support software such as compilers, debug aids, librarians, etc., must be available the first transition day to support program maintenance. Personnel must have already been trained in the operation of and be intimately familiar with the design and coding of the new programs. This is greatly facilitated if the maintenance personnel have been involved in the development process as suggested earlier.

By the time the "turnover" point is reached in a system's life cycle, corporate level management cannot do a great deal more to influence the quality of a system. There is, of course, plenty of work and management still involved, but the character of the system has been, to a great extent, cast in concrete. If the system is to be modified in the future, then that effort will be tantamount to a new development and the earlier life cycle phases must be repeated.

Finally, the configuration management of the system continues to be of great importance to corporate management. The change control process that was so critical during the integration/test phase must be continued. It is necessary to know the exact status of the software at all times. The integrated hardware and software configuration control board that should have been established early in the development process must remain in existence for the lifetime of the system.

V. Management Controls

In spite of its position, corporate management has few direct control mechanisms for enforcing policy and standards among projects. The most effective control is often referred to as "the golden rule" (he who has the gold makes the rules). Controlling the availability of funds has a very positive effect over adherence to the rules. Withholding of procurement authority until all the various rules, policies, standards, etc., have been complied with is very effective in the initial stages of a project. Beyond these two mechanisms, the remaining controls tend to

be more indirect. Examples of these are individual performance evaluations, promotions, personnel transfers, psychological methods, and leadership.

The explosive proliferation of digital technology throughout system acquisitions has served to amplify the need for astute, disciplined software management. It is tempting for headquarters to delve too deeply into a project's management. However, the implementation of objectives and controls should be constrained by the following fundamental rule: corporate management should only be concerned with telling project personnel *what* to do–not in telling them *how* they should do their individual jobs.

VI. Management Initiatives

Although the need is apparent, there appears to be precious little innovative activity in the area of software management. Perhaps this is so because computer scientists believe that management per se is not their business, and the management professionals assume that its the computer scientists' responsibility. Be that as it may, there are many software management problems yet to be solved as well as plenty of room for improvement in the way that such business is conducted. In short, there are many challenges and opportunities for initiatives in software management.

One of the greatest corporate software management needs is the development of new types of contracts for procuring software. Because of the inability to adequately specify and evaluate software, the majority of contracts used are of the "Cost Plus" type. Sometimes they contain incentives by way of awards and penalties. Unfortunately, award and penalty determinations easily can be made on the basis of a "personality contest" or a legal protest. The incentives and motivations in "Cost Plus" contracts are such that it is not in the contractor's best interest to do a good job. The poorer job that he does, the longer he is kept on the payroll. Low quality also offers the potential for new business, by either repairing the shoddy workmanship or developing a replacement system. The higher the costs, the more profit is realized. There is, thus, little incentive to be early or on time in delivering a product. Additionally, if the software is a low quality product written in assembly language, then only the developer knows how the program works. It would cost too much to pay someone else for program takeover.

More desirable would be "Fixed Price" contracts, with realistic incentives that could be determined objectively, and with the availability of warranties on software. These would direct the vendor's incentive to perform along more productive channels. Also needed is a program generation and development methodology which would minimize dependence on a given vendor's unique capability. This would promote competition and provide an avenue for increased portability of programs and systems.

There are several R&D tasks embedded in these concepts. First, the customer needs to be able to write complete, correct, and unambiguous specifications for the software he wants to acquire. Efficient application oriented specification languages would be valuable tools for this process. A good software contract needs to contain reliability and maintainability requirements. Since these concepts have not yet been defined, how then can they be included in a legal document? Assume for the moment that the foregoing could be done, how then could the software be evaluated to determine if it were in conformance with the contract specifications? Definition of error, performance criteria, software quality criteria, and sufficient testing are just a few of the tools needed to measure contractor and program performance. To assure vendor independence, the exact status and character of a program at any instant in its life cycle must be completely known. There are several activities underway that may turn out to facilitate this independence in operational environments, such as "PARNAS Modularity" [6]. The question is, are they sufficient?

The most costly corporate resource used in the development and maintenance of software is people. There are two ways of reducing manpower costs: increasing productivity and increasing automation. Much in computer science R&D is already devoted to these areas, but some solutions are needed quickly, e.g., automated verification, validation, and testing procedures [7].

Computer program documentation is, of course, required, but how to produce it and to what extent it is to be produced are subject to a lot of debate. Program design languages and self-documenting code are envisioned as solutions to how programs should be documented. But they only address programming per se. What about requirements specification, test requirements and specifications, user and operator instructions, interfacing definitions, and integration requirements? There is a pressing corporate need to minimize program documentation requirements and then generate the essential documentation set automatically. This function is presently being performed in a very labor intensive, error prone, and expensive manner.

The methodology for transitioning computer science technology from the R&D laboratory into the industry and user community is an area needing considerable effort. For example, when program verification techniques and software correctness proofs begin to approach reality, how will they be tested for feasibility for practical usage? Then how are the techniques to be injected into the developer's inventory of tools? Reflect back on how Structured Programming evolved. It has taken far too many years for it to become an accepted programming principle [8]. Management had too much at stake to gamble on a new, unproven procedure. It needed several valid implementations to convince the development community that it was a safe, cost-effective discipline. In fact, it is still not yet universally accepted throughout the software community.

VII. Summary

Corporate level software management's prospective of software development and maintenance is far more encompassing than that of the individual manager of projects and systems. Often, decisions made at this higher level appear not to be cost-effective for a given project; however, when all projects and total life cycle considerations are taken into account, they

should reduce overall software costs. Attention should be focused on those activities which have high payoff during the long maintenance phase of a system's life cycle.

Discipline (and its standardization subset) is the key to all management techniques and tools. Until recently, the software domain was largely uncontrolled. This is now changing and great opportunity exists to improve the management of software by the imposition of discipline.

Successful implementation of corporate policies which enhance the professionalism and discipline in software processes is the key to success in this area. These policies must come from the top and be enforced with the same vigor as other key corporate initiatives. There is no substitute for good management here, nor excuse for lack of it.

References

[1] F. P. Brooks, Jr., *The Mythical Man-Month*. Reading, MA: Addison-Wesley, 1975.
[2] B. W. Boehm, "Software and its impact: A quantitative assessment," *Datamation*, May 1973.
[3] S. L. Elshoff, "An analysis of some commercial PL/I programs," *IEEE Trans. Software Eng.*, vol. SE-2, pp. 113–120, June 1976.
[4] C. L. McGowan and J. R. Kelly, *Top-Down Structured Programming Techniques*. New York: Petrocelli/Charter, 1975.
[5] G. M. Weinberg, *The Psychology of Computer Programming*. Princeton, NJ: Van Nostrand Reinhold, 1971.
[6] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *CAEM*, Dec. 1972.
[7] W. C. Hetzel, *Program Test Methods*. Englewood Cliffs, NJ: Prentice-Hall, 1973.
[8] E. W. Dijkstra, "GOTO statements considered harmful," *Commun. Ass. Comput. Mach.*, vol. 11, Mar. 1968.

John D. Cooper received the B.S. degree in agriculture from the University of Missouri, MO, and the M.S. degree in computer science from the Naval Postgraduate School, Monterey, CA.

He is a Commander of the U.S. Navy and Assistant to the Director (Software Management), Computer Resource Office, Headquarters, Naval Material Command, Washington, DC. This office is responsible for establishing the policy standards, and procedures for all computer hardware and software in the Naval Material Command. He is the Configuration Manager of the CMS-2 and SPL/I programming languages and is the Navy Member of the DOD High Order Language Working Group. He is also responsible for MIL-STD-1679. He is currently on the staff of the Navy Logistics Management School where he lectures on software acquisition. His previous tour of duty was at the Fleet Combat Direction Systems Support Activity, Dam Neck, VA. During this assignment, he served as the designer and project officer for the development of software for various Naval Tactical Data Systems (NTDS).

Controlling the Software Life Cycle—The Project Management Task

WILLIAM C. CAVE AND ALAN B. SALISBURY, SENIOR MEMBER, IEEE

Abstract—**This paper describes project management methods used for controlling the life cycle of large-scale software systems deployed in multiple installations over a wide geographic area. A set of management milestones is offered along with requirements and techniques for establishing and maintaining control of the project. In particular, a quantitative measure of software quality is proposed based upon functional value, availability, and maintenance costs. Conclusions drawn are based on the study of several cases and are generally applicable to both commercial and military systems.**

Index Terms—**Life cycle, software development, software management, software quality, system management.**

Manuscript received June 15, 1977; revised December 3, 1977 and January 1, 1978.

W. C. Cave is with Prediction Systems, Inc., Manasquan, NJ 08736.
A. B. Salisbury is with the Research Directorate, National Defense University, Washington, DC 20319.

I. Introduction

THIS paper is the result of an effort to investigate project management methods which were used successfully in the development of several large software systems. As used herein, "success" implies the delivery of a quality product, on time and within budget, resulting in a high degree of user satisfaction.

A number of systems spanning a significant range of applications were examined during the course of this effort. Examples include an automated engineering design system, a stock transfer accounting system, and a real-time military command and control system. Sizes ranged from 340 000 to 700 000 bytes of object code, and assembly language examples were considered in addition to higher order languages (Fortran and Cobol). All of the systems studied had the common property of deployment at multiple geographically dispersed locations, ranging from 6 to 15 in number.

0098-5589/78/0700-0326$00.75