POST DEPLOYMENT SOFTWARE SUPPORT

Allen T. Irwin
Technical Director, Science Applications, Inc.
3655 Maguire Blvd., Suite 150
Orlando, Florida 32803

ABSTRACT

As computer based training devices proliferate, the need to maintain the software associated with the devices will cease to be an isolated need and will become a general requirement. Based on experience with tactical systems, it can be estimated that the maintenance of the software associated with a simulator over the life of the device will cost at least as much as the original acquisition cost of the software. In this paper the procedures and requirements for software maintenance are analyzed. The many associated trade-offs are examined. Based on the requirements of post deployment support the impact on the acquisition process is examined. Recommendations are made as to how the acquisition or development phase can best support post deployment support activities for software such that system utility can be maximized and life cycle costs minimized.

INTRODUCTION

In a recent IEEE tutorial on software cost estimation, Putnam shows software maintenance costs ranging from the same as initial acquisition costs to nearly three times the cost of original software development. (1) Jensen and Tonies report figures ranging from 50% to 80% of total software budgets being devoted to software maintenance. (2) Clearly, the Life Cycle Cost of a training device involving software will include significant expenditures on the support of the software after it has been deployed for operational use. Thus, proper planning for post deployment software support offers the training device purchaser at least as effective control over life cycle costs as does proper planning for the development of the software. Indeed, as the expected useful life time of a training device gets longer, more and more leverage will be available through control of post deployment costs. This paper attempts to describe the nature of post deployment software support activities, the nature of the resources necessary to accomplish the activities and to provide some indication of the planning considerations to be addressed during system acquisition.

SUPPORT ACTIVITIES

The need for post deployment changes to software come from such sources as:

- o Residual errors not previously discovered.
- o Inadequate design leading to dissatisfaction with performance.
- o Increased user sophistication leading to extension of performance requirements.
- o Modifications to the basic system leading to new training requirements.
- o Changes in tactical doctrine also leading to new training requirements.
- o Upgrading of trainer hardware leading to new interface requirements.

All of the above sources generate a continuing need to change or correct the software of a training device over its life time. The provision for accomplishment and management of these changes in a cost effective manner needs to be a high priority goal of those acquiring the training device.

The types of activity which characterize post deployment software support are illustrated in Figure 1. The two are not totally distinct and that which begins as one type of activity generally concludes as the other. The two types of activity referred to are immediate problem response and on going scheduled version production. Most training devices are of sufficient importance today, that significant error conditions, once discovered, cannot be tolerated while the lengthy process of formal version production is accomplished to correct the error. Instead, some form of immediate action is required to bring the device back to a state that is ready-for-training. Immediate action is characterized by focus on the single problem at hand and often takes the form of a "work around" that avoids the unwanted symptoms rather than curing the source problem. Such fixes are temporary in nature and are segregated from the formal change package that will eventually resolve the underlying problem.

One of the most important aspects of immediate action programming is simplicity. The programmer must find the simplest means possible to avert the unwanted performance. He must seek to minimize his impact on the system. If he does not, he will most likely introduce two or three new problems for each one he resolves. Some times he will find an obvious coding error and will totally clear up the problem with a simple correction. However, such errors will generally be found and eliminated early in the life of the system. Problems that are discovered past initial deployment are often caused by complex interactions of code that are only apparent now that the user has gained a level of sophistication in the use of the training device. For immediate action, it may be best to suppress an erroneous output rather than to make changes impacting multiple functional areas of the simulation. Very often the instructor can easily supply the missing information to the student allowing training to continue.

Meanwhile, thorough design analysis of the reported problem is conducted and a complete correction is developed, implemented and tested. Once that process is complete, the new version of the software, containing corrections to multiple problems and design enhancements as well, is released to the field. The process by which a problem report is processed to final solution requires formal control as described in the following section.

Processing of a Problem Report

A problem report is initiated when user of a training device sees some feature that does not appear correct to him. The report is forwarded through normal maintenance channels until it reaches the support facility

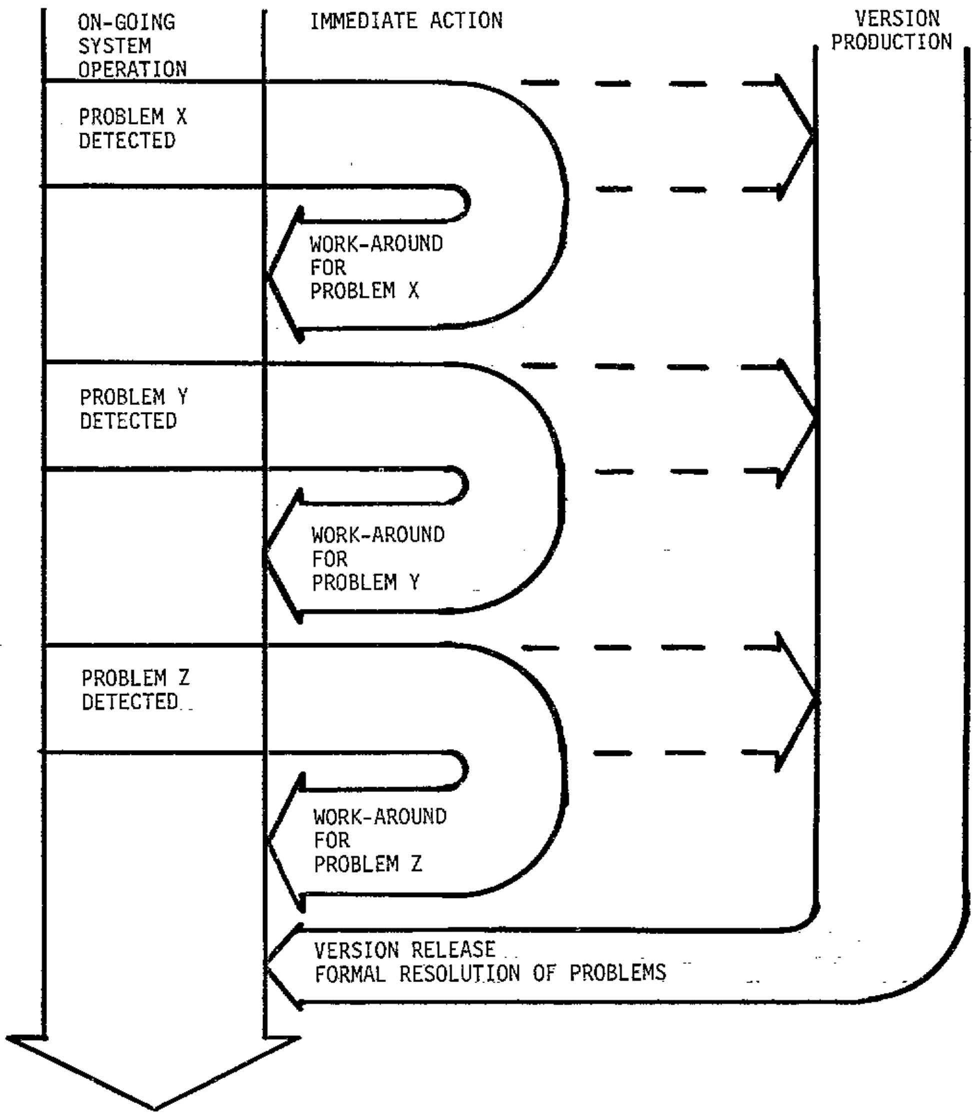
ON-GOING SYSTEM OPERATION
IMMEDIATE ACTION
VERSION PRODUCTION
PROBLEM X DETECTED
WORK-AROUND FOR PROBLEM X
PROBLEM Y DETECTED
WORK-AROUND FOR PROBLEM Y
PROBLEM Z DETECTED
WORK-AROUND FOR PROBLEM Z
VERSION RELEASE
FORMAL RESOLUTION OF PROBLEMS

FIGURE 1 - POST DEPLOYMENT SOFTWARE SUPPORT ACTIVITIES

where an initial analysis is performed.

The initial analysis is technical in nature and seeks to answer the following questions:

1) Can the situation described in the problem report be duplicated? If not, the user reporting the problem is queried for more information until the problem can be repeated or is traced to improper operator action.

2) Is the system performing in accordance with specifications? If system performance does not match specified requirements, corrective action is explored.

3) If the system is performing according to specification, does the problem report indicate a situation that requires a design change? Very often an apparent problem, particularly in a simulation is, in fact, a trade-off that was made to gain some more important area of performance. For example, a simulation may have sacrificed full fidelity of some feature to maintain real-time operation. Such past decisions must be a matter of record to avoid unnecessary analysis each time that a new user detects the apparent discrepancy. On the other hand, continued reports of such a situation may be the basis for a re-evaluation of the design, indicating that the apparent training impact may have been more severe than originally anticipated.

The initial analysis, therefore, establishes if a problem really exists, and if it does, whether its *resolution requires only corrective action or if it requires* a design change.

Once the initial analysis has established the required resolution, the necessary corrective action or design change is scheduled for implementation in a forthcoming version. Based on the required effort, the urgency of the change and the available resources, one of the upcoming version releases will be selected for incorporation of the new code. If a minor correction of a few instructions is all that is needed, it will most likely be incorporated into the next release. However, each version has a "freeze date" beyond which, no further modifications can be incorporated. Actions that require significant design effort may be scheduled for further analysis before actually being scheduled for implementation. An important aspect of the analysis at this time is to establish "user" concurrence on the proposed design change. Only after the user has validated the new requirement can it be scheduled for incorporation in the system.

Prior to scheduling for implementation, any design change must be analysed to develop a proposed approach, a probable cost and an expected impact on system performance. This is, of course, standard ECP procedure. The process leading to formal scheduling for implementation is illustrated in Figure 2.

The "Version" Concept

Systems containing relatively little software require only *informal means of control. The trend today,* however, is toward large, complex software. Such systems require formalized baseline control and disciplined change procedures. Formal procedures become necessary whenever the software is too large or complex to be maintained by one or two people. Formal control is also needed when the software is one system of several being maintained by the same people at a central facility.

The primary means of obtaining formal control over software has been by means of Version Releases. Periodically, over the operational life of the system, new versions of the software are released for operational use. The frequency of version releases varies with the criticality of the system, the maturity of the software, and the dynamics of the system.

Critical systems, such as operational air defense systems may generate frequent version releases to insure continued peak performance of the system. As software matures, fewer and fewer latent errors are present, and version releases can be further apart without adversely impacting the system. Some systems will always be highly dynamic. Simulations of new aircraft, for example, will be in a constant state of change until the aircraft they represent mature and stabilize into a fixed version. For most systems, software version releases will be frequent during initial operations and will taper off to an as-needed basis as the system matures.

Just what is a version? Since it is released for operational use, it obviously must have undergone all the controls and checks and tests that any new item must undergo before it is released to the field for operational use. *A software version must be thoroughly validated* for correctness of design and implementation as well as for correctness of technical performance before it can be released. Further, all ncessary items needed to support the version must be validated and ready for simultaneous distribution to the field. This includes all user's manuals, performance aids, technical documentation, maintenance procedures and instructions and any necessary training materials. In situations where a significant design change is being implemented, training teams must be prepared to install the new version at each site and to train the operational crews on the use of the new system. As can be seen, support of systems at multiple sites poses a logistical problem for software as well as for hardware.

Version Production

Version production is the entire process of analysis, planning, design, implementation and test necessary to convert selected modification requirements into a software version ready for release to the using units. This process is illustrated in Figure 3. To accomplish version production, all selected changes must be carried through a design phase with appropriate reviews and coordination. The latter is particularly important in software maintenance efforts. The tendancy to view a particular change or enhancement as an isolated problem makes the possibility of inadvertant adverse effects on other areas of performance an ever present danger. A thorough, formal and disciplined process of review and coordination must be established and enforced throughout the version production process. If not, the version will never survive the prerelease testing to qualify it for operational use.

Design Teams

One technique for accomplishing such coordination and review is the establishment of design teams. Each change or design requirement is assigned to a lead analyst. Then a representative of each functional area is appointed to form a design team. Each member may well serve on several design teams, because their primary function is to review the work of the lead analyst to ensure *that impact on their functional area of* responsibility is correctly understood. The lead analyst is, of course, selected from the functional area most severely affected by the change. It is his responsibility to coordinate his design with all members of his team and to obtain formal concurrence from the team that this design is an acceptable implementation for all functional areas.

The design team also serves to coordinate the documentation and testing associated with the particular change package. Changes to user manuals, technical documentation, maintenance manuals, and training materials must all be prepared and coordinated. Tests must be prepared that will test out the change itself and also to validate that all other areas are not impaired.

Version Testing

Version testing mirrors normal software development testing, *but with some significant differences. For* modifications representing new design, a bottom up approach is generally followed. The modified modules are tested and debugged in isolation then integrated into the system and tested for correct operation at the system level. Smaller corrections may only require verification at the system level to show that they have, in fact, been correctly implemented. The most significant fact about version testing is that any one version changes only a very small percentage of the system software performance requirement. Thus, system level version testing does not require a new system test for each version, but rather a new version of the already validated system test. This approach to testing can allow significant savings in the testing effort required to validate a new version for release. A very desirable approach is to develop an automated system test that provides stimulating inputs *and automatic analysis of results (See Figure 4). Such a* test tool, once accepted by the user can greatly reduce the effort of version qualification. The automated test program can short circuit normal operator input and output since it is required to show only that unmodified software still performs correctly. Thus, by running prerecorded inputs through the system and automatically comparing the outputs with expected results, it validates the operation of the software not affected by the changes incorporated by the new version. (The system test itself,

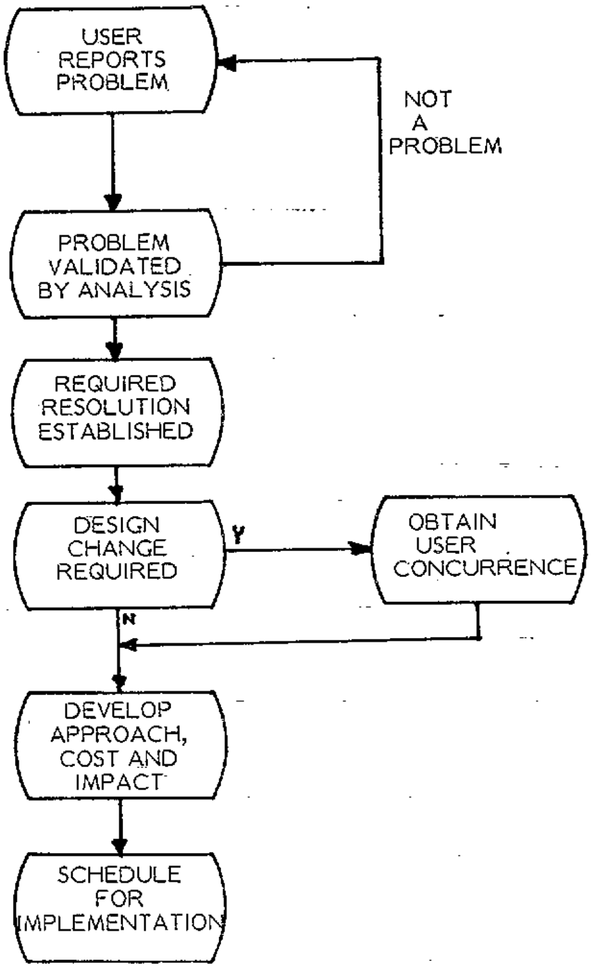
USER REPORTS PROBLEM
NOT A PROBLEM
PROBLEM VALIDATED BY ANALYSIS
REQUIRED RESOLUTION ESTABLISHED
DESIGN CHANGE REQUIRED
Y
OBTAIN USER CONCURRENCE
N
DEVELOP APPROACH, COST AND IMPACT
SCHEDULE FOR IMPLEMENTATION

FIGURE 2 - PROBLEM REPORT PROCESSING

must be updated for each version, however, this is much more manageable than developing a new test each time.)

The test cycle for the version is as follows:

- o Debugging at the module level
- o Testing of new features for correct implementation at the system level
- o Validation of unmodified software using the automated system test
- o User (Operational) Testing of new features

An important consideration of the last step in the above sequence is how much user testing is required? Clearly, the user of the training device must be satisfied that it is performing correctly. However, it should not be necessary to revalidate by operational testing all performance parameters of the system at every version release. The ability to satisfy the user (he is, after all, the customer) that he needs only test the new features of the version will depend heavily on the degree to which the user can be convinced that the automated version test does successfully validate the unchanged software. This is further support for a well planned and well documented system test.

Formal, centralized control is a necessity in the test program. If central control is not maintained, testing can degenerate into an unending set of failures as test after test encounters unexpected side effects. This can only be avoided if proper resources are committed to the testing effort throughout version production. Testing is also very sensitive to "Version Freeze Date". If the input of change

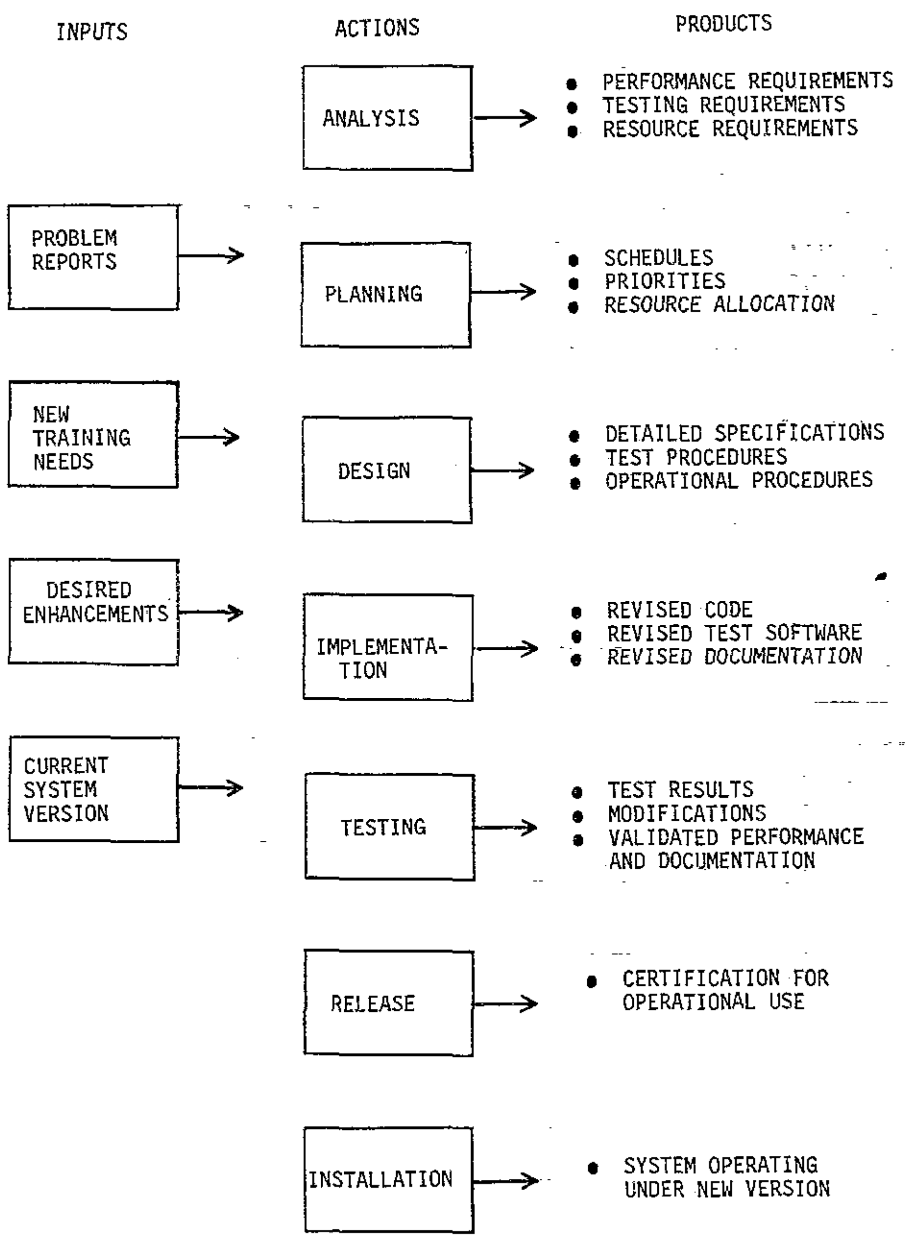

FIGURE 3 – THE VERSION PRODUCTION PROCESS

requirement for a particular version are not cutoff early enough, testing will suffer along with design. This can be offset by scheduling version release dates with sufficient frequency to reduce the pressure to keep adding "just one more change".

Scheduling of Version Releases

The time required to design, develop, implement, test and release a version will necessarily vary with its contents. If the version is primarily to correct coding errors then it will require a short period to prepare it for release. As new design effort is added to the version's contents, the required time length will go up. Thus, version release may take from 6 to 18 months. When significant hardware changes are also involved, the release time could easily approach the initial development schedule and run as much as 3 years.

For administrative reasons, it may prove desirable to schedule versions for release on a fixed timetable. In this case, version content can be adjusted to meet the schedule. In addition the start date for versions selected for significant change efforts can be set ahead of the normal version start time. The phased nature of version production lends itself to overlapped production of versions. That is, while one version is undergoing testing by the testers, a second one can be in the hands of the programmers, while system analysts have begun the design of a third version. In this manner three versions are in production at any one time. Version release occurs more frequently and the technical staff is allowed to specialize in the three specific areas of design,

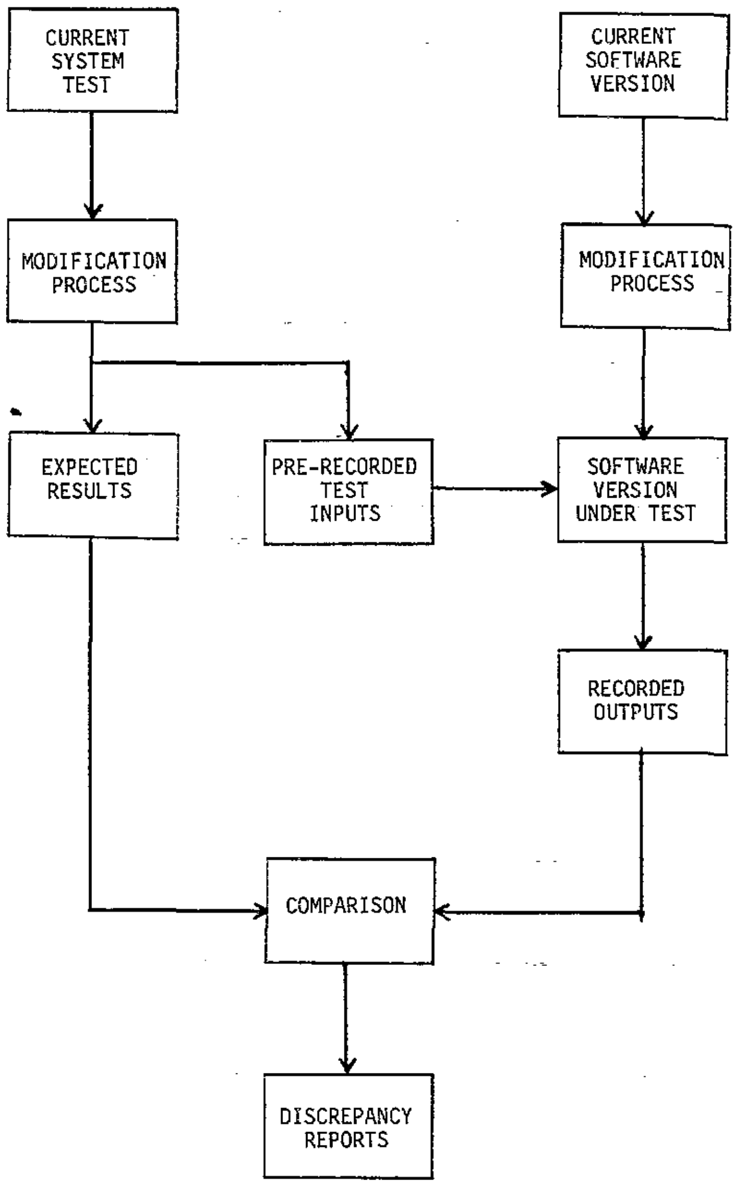

FIGURE 4 - AUTOMATED TESTING PROCESS

implementation and testing. Such specialization leads to increased efficiency since each area requires particular skills and not many individuals will be equally capable at all three areas.

PLANNING FOR POST DEPLOYMENT SOFTWARE SUPPORT

In establishing the Post Deployment Support Concept for a training device, questions such as where software support will be performed, who will accomplish the support, and what equipment will be required must be answered. The answers to such questions are not arbitrary, but are determined by several contributing factors.

Locating the Software Support Capability

Assuming that we are talking about a training device with a significant software support requirement, then the issue of where to perform the support task is largely a question of resources. Perhaps one of the primary considerations is the availability of equipment.

It does no good to locate the maintenance facility at the training site if the training device is so heavily committed to training use, that no time is available for software maintenance usage. The temptation to save money by planning to do all software maintenance on operational equipment must be balanced against the very real cost penalties of having programmers being paid to sit around waiting for computer time. The early planning for training devices generally includes an assessment of device utilization. Using this assessment plus an estimate of how much device time will be required for hardware maintenance, provides an indication of what time will be left over for software maintenance utilization. Analysis of the software maintenance task, based on the dynamics of the system that will lead to changes to the training device can provide an estimate of the size of the software workload and hence the required device access. If a conflict exists as to device availability, then the thought of using operational equipment should not be carried further since the initial estimates of software change activity usually tend to be low!

Once the decision is made to acquire a software support facility (i.e., computer equipment dedicated to software maintenance) the question of where to locate it still must be resolved. At this point the other essential resource must be considered--personnel. If a centralized support facility already supporting similar or related training devices exists, then it provides an attractive base for economically expanding to provide support for the new system. If a stand-alone maintenance facility is to be established, it will generally require more personnel than would be required as add-ons to an existing facility. An existing facility also provides a source of personnel experienced in the technical and managerial skills associated with software maintenance. (It should be recognized here and now that the developing contractor is not going to turn over his design and programming staff to maintain the delivered equipment. They will be off designing and programming new products.) A brand new software support facility will always undergo an expensive learning curve as the new personnel come up to speed with the system and with the maintenance process.

The equipment placed in the support facility need not necessarily be as elaborate as the full up training device. The majority of software development and testing can be accomplished without the full set of hardware required for the more elaborate training devices. While training time will not usually permit software maintenance to be fully supported on operational equiment, it usually can support some of the final system testing necessary for version release. Thus, a concept of shared facilities may prove a useful compromise with the majority of software maintenance being accomplished at a less elaborately equipped support facility and the formal system level version acceptance testing being accomplished at a designated operational "test" site.

When choosing the operational site or an offsite facility location, consideration should be given to the ease with which qualified personnel can be recruited andretained at the locations under consideration. Questions such as the availability of trained personnel, and the desirability of the geographical area will have long term significant economic impact on the operation of the facility.

The "WHO" of Software Support

Daley, speaking of commercial software, estimates that one full time programmer can maintain 10,000 lines of realtime software or 30,000 lines of support software. (3) This figure does not include configuration management, supervision, field support or other overhead. While his estimate is not necessarily directly applicable to training device software, it does serve to illustrate that the manpower requirements to maintain any significant software package are not small.

As Daley also points out, a programmer will be more effective in a maintenance roll if half of his time is devoted to maintenance and half to development of new software. (3) This effect derives partly from motivation and partly from proficiency maintenance gained by experience with new systems. This factor seems to argue against dedicated software support facilities. However, the experience of developing new software can be provided at a support facility if system enhancements are also assigned to the facility for development. Otherwise, it can be expected that a facility devoted solely to error correction and minor enhancements will suffer as good people move on to more interesting (and better paying) work and those that remain lose touch with the developmental side of software programming.

The question of contractor support versus Government in-house support of software is frequently raised. There is, however, no simple one time answer to the issue. The choice must be made and justified on a system by system basis.

The key issue, if Government in-house support is being considered very often comes down to whether or not the Government can make available the personnel resources needed to maintain the software. Right now the Government has limited capabilities in this area, however, as tactical computer systems proliferate, the software support capabilities within DoD will expand dramatically. Thus, it may be that in a few years, assignment of training device software to a Government software support facility will be routine.

The possibility of Government support of the software at some future date is another argument for the acquisition and maintenance of thorough and complete documentation of <u>all</u> software, even if it is to be initially supported by the developing contractor.

Post Deployment Software Support Equipment

When acquiring a training device, considerable thought should be given to the acquisition of software support equipment. As mentioned above, operational equipment is often not available for software support. This makes it necessary to have computer equipment dedicated to the support role. One means of accomplishing this is to assign prototype equipment to the support role upon completion of prototype testing. Another means is to buy excess computer power so that both operational and support functions can be accommodated in a timesharing mode. (This is only possible where the operational requirements can tolerate the limitations of time sharing operation.) In many cases it will be necessary to buy (or lease) a totally separate set of equipment for the support role.

Whichever method is selected, it is important that requirements for the support function are specified and designed in from the beginning. Attempts to go back and force such functions in later on always meet with limited success and are generally much more costly than when they are recognized and designed in from the beginning.

When the prototype of a training device is to end up as the software support facility, thought should be given to procuring excess computer power in the prototype configuration. Not only will this make the development of the operational software easier, it can also allow the support facility to support multiple programmers in a time-sharing mode. It will also make it possible to establish a test environment for the operational software by the addition of other software designed to provide on-line analysis of operational software performance.

SUMMARY

In conclusion, the planners of a training device development can, through proper provisions for post deployment software support, do as much or more to control the overall cost of the training device as can be accomplished by proper provision for the initial software development. Coversely, failure to consider support requirements for software can be just as disasterous as failure to consider hardware support requirements. Lack of adequate provision for software support can prevent effective use of the training device and can cause excessive growth of life cycle costs.

Software support, once the training device is deployed, will consist of software modifications to enhance device performance, to eliminate latent errors, or to reflect changes in the operational equipment.

These modifications must be managed to ensure that the operational validity of the training device is not lost. Thus, formal procedures must be established for the post deployment support operations.

Two primary activities characterize post deployment support operations; immediate action and version production. Immediate action is intended to provide quick fixes or work-arounds for problems detected during system operation. Version production is the formal process of analysis, planning, design, implementation, testing, release and installation that provides integrated and validated software enhancements and fault corrections that ensure effective and efficient use of system resources throughout its life cycle.

To make certain that the support facilities and resources required for post deployment software support are ready and in place when needed, planning must begin with the initial training device concept formulation. To accomplish the necessary planning, consideration must be given to where the software support activity will be accomplished; what personnel resources will be, a) required, and b) available to provide the support; What equipment is to be used for post deployment support and how it is to be obtained, and how the overall effort is to be managed.

It is strongly recommended that a Post Deployment Software Support Plan be developed early in the acquisition process and that it be maintained up-to-date throughout the device life cycle.

BIBLIOGRAPHY

(1) Putnam, Lawrence H., Software Cost Estimating and Life Cycle Control, IEEE Tutorial, Computer Society Press, 1980, pages 13ff.

(2) Jensen, Randall W. and Tonies, Charles C., Software Engineering, Prentice-Hall Inc., 1979, pages 403 ff.

(3) Daley, Edmund B.; Management of Software Development, IEEE Transactions on Software Engineering, May 1977, page 232.

ABOUT THE AUTHOR

Mr. Allen T. Irwin, Technical Director, Education and Training Technology Division, Science Applications, Inc., has provided technical support to PM TRADE for over 3 years. His previous experience includes software acquisition support to other Army Projects, development of software for the US Air Force and software maintenance activities with the SAGE and BUIC air defense systems.