# SOFTWARE QUALITY ASSURANCE APPLIED TO TRAINER SYSTEM DEVELOPMENT

Terry Tierney, Senior Quality Assurance Engineer
Link Division, The Singer Company
Binghamton, New York 13902
(607) 772-3011

## ABSTRACT

This paper will discuss the application of software quality assurance techniques to trainer software development, taking into consideration military standards and specifications and the unique characteristics of trainer development programs. Because military customers are paying greater attention to software development and software documentation, software quality assurance has become an increasingly important management objective. Although there are no quick recipes for adapting software quality assurance techniques and standards to trainer development, this objective can be met by analyzing the software specifications and standards along with the software quality assurance specifications and standards, by considering the unique aspects of trainer development, and by considering the aspects of applying and adapting software standards to trainer development. First, the various specifications and standards that apply to software development must be analyzed with particular attention paid to their interrelationships and to their relationships with deliverable data items. Then, this conceptual framework must be related to the software quality assurance standards and specifications. Differences and similarities between the standards and specifications written by different military customers will also be considered. Given this overall picture of the requirements for software development and quality assurance, the unique aspects of trainer development may be considered. Among these are shortened schedules, abbreviated data requirements and the application of weapon system standards to trainer development. Once the various requirements and the peculiar constraints of trainer development have been analyzed, the next step is to consider the meaningful application of software standards and quality assurance techniques to trainers. Among these considerations are cost effectiveness, who should accomplish the various quality assurance tasks, applicability of internal standards, whether tasks are best handled on a company or program basis, tailoring quality assurance functions to program needs, and the problem of assuring quality of software when there are no specific software data item requirements. In conclusion, this paper will present an approach to developing a software quality assurance program for trainer system development.

Software management and control presents the same challenge to the trainer industry that it does to major computer and weapons system manufacturers. Although the overall development cost for a training system may be only a fraction of the cost for the system it simulates, the percentage of development cost allocated to software may be greater in the training system because the trainer industry has become increasingly software intensive and there is every indication that this trend will continue. Given the fact that the cost of software development comprises a larger portion of the cost of system development each year, controlling software development has become a primary management concern.

Methods and procedures for control of cost and quality of hardware development have been applied in industry for so long that they have become almost second nature and we no longer ask whether or not procedures such as an engineering drawing system or a hardware change control system should be instituted. Now that control of software development has become more of a concern, we cannot treat the software portion of system development as a black box design for which we dedicate money and manpower at the beginning of a program, only to have it slip from view until test and integration when it either works or does not work. But it is obvious that the same controls that have been applied to hardware development cannot be applied directly to software development and also that controls applied to major system software development might not be feasible or cost effective when applied to trainer system software development.

Control of software development is generally divided into management functions and quality assurance functions. Software management includes functions intended to specify and control the design, and software QA includes functions intended to monitor the design process according to the management objectives. A number of software management and QA techniques and procedures have been developed which can provide models for trainer software management and QA, and provide techniques that may be adapted to fit the needs of a trainer development program. Since the government is the major buyer and user of software, it stands to reason that the control of software design would be a major government concern. This concern

is reflected in military standards and specifications, which include a rather complete blueprint for software development. It should be noted that the software management and QA techniques employed by many individual companies are either derived from or reflected in these standards. This is not to say that the military standards are the last word in software development or that they are unilaterally complete or effective, but they are a useful source for companies desiring to institute software management and control techniques and an effective software QA effort.

The military standards that describe the software development effort are MIL-STD-483, MIL-STD-490 and MIL-STD-1521, and documents which are devoted primarily to software QA are MIL-S-52779, MIL-STD-1679 and MIL-STD-1644. Briefly, MIL-STD-483 can be seen as the grandfather of this group of documents because it details the content requirements for software documentation along with MIL-STD-490. MIL-STD-1521 specifies the documentation required at the completion of the various phases of software development. MIL-S-52779 calls for the implementation of a software QA program and could be described as a parallel document of MIL-Q-9858, which specifies quality control requirements for hardware development. MIL-STD-1679, a Navy document, includes more detailed software quality program requirements and references a set of software documentation which is similar to that described by MIL-STD-483. Finally, MIL-STD-1644 is basically a MIL-STD-1679 approach applied to trainers with very little tailoring.

Two basic concepts are common to all of these documents and these two concepts can be applied to form the basis for a trainer software management and QA effort. The first of these is the concept of development phases. That is, the software development process is divided into successive phases marked by milestones, at which the design up to that point is analyzed and validated. The second concept comes into play in the form of documentation. Each milestone is represented by a documentation set which may consist of a specification or flow charts or printouts, depending upon the particular phase that is represented. This documentation must be seen as a management tool which provides visibility of the design and which provides sufficient detail to validate the design. The documentation also specifies what will be accomplished in the next phase so that at the completion of each phase, the documentation may be validated by comparing it with the documentation generated in the previous phase. This provides for traceability of the design to the original system requirements throughout all phases of development. A simplified schedule for a typical software development effort is shown in Figure 1.

For the purposes of this discussion, the software development process will be divided into four major phases: Analysis, Design, Coding, and Test and Integration. Other divisions are possible, provided each phase leads to a milestone which can be seen as a design baseline represented by a specific set of documentation that describes the design up to that point. The terms used to describe the documentation will be taken from MIL-STD-483, recognizing that the MIL-STD-1679 differs primarily in form, rather than intent.

If we envision a software development effort beginning with a decision to allocate funds to a certain project or beginning with a contract award, the first task is to analyze the system software requirements for testability, functional grouping, and implementation. Testability is a determination of whether or not the system requirements can be implemented and tested, because once the design is complete we must be able to prove that all system requirements have been met. Also during the Analysis phase, the system requirements might be divided into two or more functional areas, often called Computer Program Configuration Items, if it appears smaller units of design may be easier to control than one larger system. The major activity of this phase is the translation of system requirements into software requirements. That is, we analyze the overall requirements and come up with a set of instructions that can be understood and followed by a software designer. These instructions are contained in a Computer Design Specification, or a similar document, for each Computer Program Configuration Item. This document will include such information as block diagrams, general flow diagrams, interface definitions, and software system requirements. Often the milestone which closes the Analysis phase is the Preliminary Design Review, during which the Computer Program Design Specifications are compared to the overall system requirements and all necessary changes are incorporated.

During the Design phase, the software designers follow the requirements of the design specification and draft the Computer Program Product Specification. Among the activities in this phase are the preparation of detailed flow charts, developing a top-down module structure, and allocating the available time and memory. This phase culminates in a Critical Design Review during which the Computer Program Product Specifications are compared to the design specifications and validated, once all changes are made. The Computer Program Product Specification is a draft document at this point because it will be updated to include listings once the Coding and Test phases are complete.

Following the Critical Design Review, the product specifications may be turned over to the programmers for coding. Completion of this effort results in a milestone review which may take the form of a design review or a similar activity, such as a code walk-through. The purpose of this milestone is to check the code against the approved flow charts contained in the product specification before the code is entered into the computer load. This may be done on a module by module basis or in larger units. The documents that describe this milestone are the product specifications, which are updated to include the approved code.

A parallel branch of the development effort which has not been mentioned so far is the development of test plans, test procedures and test software. During the Analysis phase a plan to test the completed software should be established and the necessary procedures written. Any test and analysis software that will be required to verify system requirements should be designed and documented in the same way that the rest of the system software is designed. That is, test soft-

ware should not be a last minute consideration. It may be helpful to allocate a Computer Program Configuration Item to the test software during the Analysis phase to ensure that it will be ready and operable when testing begins. Test plans and procedures should include provisions to test every requirement of the system specification.

The Test and Integration phase usually begins with some form of module testing. Modules may be verified against the design and product specifications according to the approved test plan and procedures. Verified modules are compiled into their respective Computer Program Components and further testing is conducted to verify the entire component before it is compiled into the Computer Program Configuration Item. The complete configuration item may then be validated against the design and product specifications. Once we have verified the software load, integration with the hardware may take place, and this process culminates in a testing procedure which validates the entire system against the system specification. Following successful completion of all testing, the updated product specifications define the production baseline for the computer program system.

So, we began with the system specification, gleaned from that document the software objectives, defined specific software design tasks, divided the design effort into phases that correspond to visible baselines, which were themselves defined by a specific set of documentation, and then compiled the software packages into a complete system which we verified against the system specification.

The objective of the software QA effort is to monitor this process and ensure that the system requirements will be met. Again using the military specifications and standards as a model, the software QA program tasks will include documentation reviews, participation in design reviews and walk-throughs, independent audits of design procedures, corrective action and trend analysis, and configuration control. Included in the configuration control tasks are monitoring of library controls, review of test reports, and change control.

In general, there are four design characteristics that the software QA representatives will hope to identify during the QA reviews and audits. The first of these characteristics is traceability of design requirements, which means simply that at each milestone of the design process, we should be able to trace each element in the software documentation back through the product specification and the design specification to the system requirements. If the documentation has been completed correctly and if the design has been documented completely, management confidence in the progress of the design will be gained from the reviews and audits. If design traceability has been lost, this problem will be evident in reviews and audits and there will be less confidence that the design will eventually meet all system requirements. Secondly, the various QA reviews and audits will determine whether or not all interfaces between software elements and between software and hardware have been identified and properly defined. Again, this is a major QA concern because if interface definition is ignored in the Analysis and Design phases of the development effort, there may be difficulties when we try to compile and test the software. The third consideration is to ensure that the design is proceeding according to the baselines established at the previous development milestone. That is, we check the design specifications against the system requirements, the product specification against the design specification, and the coding against the product specification. Finally, the reviews and audits will also ensure that the documentation conforms to the established standards for format and content.

The QA tasks and the development process we have briefly outlined in this paper are intended primarily for large scale design efforts and must be tailored to meet the specific problems often encountered in trainer system design efforts. Characteristic problems of trainer system development are shortened schedules, associated data requirements, and the application of specifications more suitable to development of critical systems or larger systems than to development of trainer systems. Managers and engineers experienced in trainer design may have asked questions similar to these: How can we write design specifications before the preliminary design review when it is scheduled two months after contract award? Is it cost effective to write design specifications if they are not required by the contract? How much of the software control exercised for critical systems is cost effective when applied to trainers?

Often trainer development schedules are shortened, especially when the trainer is intended to support a new device, because trainer design cannot begin until the device itself is designed because essential design data must be available, but the customer wants the trainer delivered in time to support training of personnel to use the new device. The optimum amount of time required to design a trainer is therefore squeezed into a shorter time frame. If schedules become shortened to the extent that tasks essential to the software design process may be limited or deleted, the impact on cost and delivery must be assessed. For example, a suggestion to skip the writing of design specifications or to eliminate module testing must be evaluated against the cost which may be incurred because traceability of the design was lost or some critical modules were poorly designed and not checked. Conversely, the cost and schedule risk of having to redesign software during customer acceptance testing may justify allowing more time at the beginning of the program to complete all design phases and QA tasks. It should be a function of the Analysis phase to determine whether contract requirements are consistent with schedule constraints, and, if there are conflicts, resolve them with the customer early in the program. A more risky approach would be to begin software development, that is, requirement analysis and the writing of design specifications, prior to contract award. In any case, if schedule brevity is a problem, it must be addressed to ensure that the software will be designed correctly and on schedule. In general, software design tasks and software QA tasks should

not be eliminated because design deficiencies that are not discovered early in the design process are more costly to correct during the test phase.

Another problem which might be unique to trainer system development is abbreviated data requirements. What is meant by this is that the customer often does not order a complete set of software documentation or will specify documentation that attempts to combine two or more functions into a single document. An example of the first case is a contract which requires Computer Program Product Specifications but does not require design specifications. Or, rather than ordering both design and product specifications, the customer may order a document that combines the design and product specifications into one specification. In either instance, what might have been an attempt to save money by ordering one less document does not save the trainer developer a penny. The reason for this is that to develop reliable software, the trainer system contractor to write both specifications. For example, if a contract calls only for final documentation, the management and QA objectives which are dependent on the visiblity provided by design documentation will be difficult, if not impossible, to meet. Rather than risk losing control of the software design, the contractor will develop some means of assuring the quality of the design and often this concern will result in some form of preliminary documentation which can be used to evaluate the design before it is released to programmers and finally to coding. This implies, of course, that in these instances the contractor will generate documentation for which he will not be directly paid. The alternative approach would be the "big bang" theory whereby the design is minimumly controlled, if at all, and the entire software system is loaded and debugged during the test phase and the final documentation is generated sometime later. Although this seems to be a relatively inexpensive approach compared to writing specifications or similar documents according to defined design milestones, studies have shown that this is not the case. In fact, it has been concern over the high cost of test and integration that has led many companies to develop software management and QA procedures that emphasize maintaining control and visibility of the software during the entire design process.

Instead of leaving the question of software QA to the discretion of the manufacturer, the trainer customer might decide to impose software QA requirements. Often, however, the requirements imposed on the training system will be taken directly from the requirements imposed on the device that the trainer will be designed to simulate. This may result in what could be termed a software QA overkill. For example, the customer might desire a training system that corresponds to a new fighter aircraft, which, because of the criticality of the system, production concerns, and perhaps the handling of classified data, was developed under very strict software management guidelines. Usually, these take the form of more documentation, more design milestones, more testing, and more record keeping. The training system, however, will probably be designed to operate in a classroom environment rather than at supersonic speeds in the upper atmosphere, will involve a much lower production run, and will process a smaller amount of classified data, if it processes any at all. To impose weapon system standards on the training system would not seem to be cost effective in this case, and could result in a situation where the cost of control exceeds the cost of the software itself. What is needed in instances such as these is a careful analysis of the cost and objectives of the software system to determine the level of control necessary to ensure that management concerns will be satisfied through all phases of system development. Hopefully, a dialogue between the contractor and the buyer will result in an approach that balances software QA cost and software performance for a given training system.

It is clear from our discussion of some of the problems encountered in training system development that software management and QA programs and techniques proven in other areas of industry must often be tailored to meet the needs of trainer manufacturers. One way to approach the question of software management and QA from a company standpoint would be to come up with a suitable in-house program. In other words, rather than simply responding to whatever software management and QA requirements are imposed by prospective buyers, the trainer contractor could develop an in-house program that fulfills the objectives of software management. The advantage of this approach is that once the in-house software management and QA programs are off the ground, unique customer requirements may only imply a tweaking of the working system rather than the writing of wholly new company procedures. Also, if the company program is structured to meet the intent of military software standards, the contractor will probably not have to revise any of the internal procedures to meet the software management and QA program requirements of most military and commercial customers.

Assuming a decision is made to go ahead with developing company software management and QA programs, there are a number of considerations which must be addressed according to peculiarities of each company and its respective business. Among these considerations are what existing model can provide the basis for the company programs, how much control should be exercised, who should be assigned to the various tasks, and whether software management and QA should be handled on a company-wide or program by program basis.

As we have already implied in this discussion, the existing military standards and specifications probably provide the best model for developing software management and QA programs because they have already been applied in many areas of industry and because by applying such a model, we could satisfy both military and commercial customers. Since much of the instructive literature on the subject of software management and QA addresses the two part specification approach as a management tool and the guidelins of MIL-S-52779 as a QA model, if a company decides to use this approach or a derivative approach, there are plenty of helpful hints available. Such an approach is also consistent with the spirit of standardization because if all contractors developed unique methods of software design and documentation, communication would be difficult.

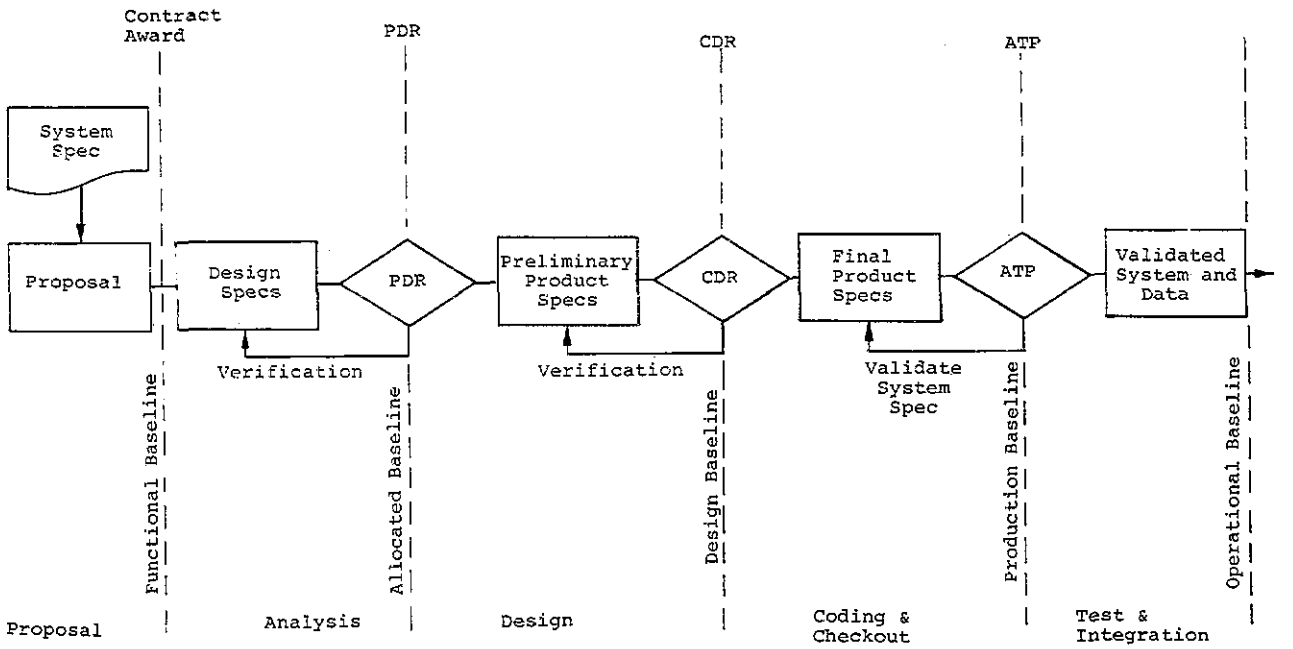
Also, it is much easier to explain your approach to a customer if it sounds familiar than if it is a unique system. Another source for software management and QA models would be the systems employed by other industries, of which there are also numerous published descriptions. In many cases, however, the titles used to describe the various design phases and milestone documents vary, but the general concepts of software management are very similar and resemble the military model.

Whatever model is chosen, a decision must be made to determine how much control should be exercised given the cost of control and the level of performance desired. In general, the software development procedures must provide enough visibility of the design that the software QA reviews and audits can be accomplished. Requirement traceability, interface definition, configuration control, and documentation standards are elements which should be part of any software management program and, as a minimum, the software QA program should include sufficient reviews and audits to ensure that these elements are present in each design. Documentation reviews can be used to check traceability, make sure that all interfaces are defined, and verify that the design is complete. Participation of the software QA representative in design reviews will provide an opportunity for questioning unclear aspects of the design. Software library controls and periodic audits of test reports and software change notice incorporaton can be used to verify that configuration control is maintained. Since it may not be feasible to review each software document in detail, sampling inspections may be planned to ensure that the established design procedures are being followed. Reviews of corrective action and analysis of error trends are useful in that they provide a basis for determining the effectiveness of software management efforts and recommending changes to standard procedures. Trend analysis, especially if a dollar figure can be applied, can prove valuable for the software QA effort because the effectiveness of such a program would then be expressed in dollars saved. This analysis could also show how well or how poorly the company software management procedures compare to published studies.

While making a determination of how much control should be exercised, it might become evident that we already have more control of software development than we realize. For example, we could have determined that the software design effort was out of control because no documentation requirements existed in the company, but found that most of the designers kept some form of notebook anyway. This would not be an optimum situation from a management or software quality point of view, but it is better than if nothing was written down at all. Perhaps the lead engineers had taken it upon themselves to insist that all designers keep notebooks and thus filled a void they recognized. What we are implying here is that all software QA functions do not necessarily have to be accomplished by dedicated QA personnel as long as all the necessary functions are assigned to someone. In many companies a software QA department has been created and staffed with software QA experts, while in other companies certain tasks are left entirely to other departments, such as engineering or configuration management, or assigned to other departments and then monitored by the software QA representatives. Library controls, for example, might be the responsibility of the software engineering department but subjected to periodic QA audits. Or, all configuration management functions, such as software change control, might be assigned to a separate department entirely and this department would also be responsible for configuration reviews and audits. It should be pointed out that MIL-S-52779 does not specify that all QA functions be performed by a QA department, provided all functions are performed.

Whether software QA functions are performed on a company-wide or a program by program basis is also left to the discretion of the contractor. This determination depends in part upon how much control is desired for systems that have no software QA or management requirements imposed by the customer. Generally, it would seem desirable to have a standard set of procedures to fall back on if none were imposed. Some of the functions, however, could be handled on a program by program basis if this approach seemed most cost effective. For example, rather than establishing a central library for the entire company, individual software libraries could be controlled by each program office. In this case we would have a company procedure which would assign responsibility for library control to the program offices. The same could be done for all other software QA functions and there could even be a person assigned to all these tasks from within the program office. Of course, during schedule squeezes, the loyalty of the QA representative to the program office might compromise the objectives we set out to accomplish. For every function assigned outside the QA department, there is a corresponding risk that the individual performing a quality task is likely to please whoever signs his paycheck, whether it be QA or the program office. Whatever approach is decided upon, it is important that it be documented in a set of established procedures so that all personnel involved in the software development effort will know their assigned responsiblities.

Obviously, there are no solutions which will work for all companies across all training system development projects. It is up to each manufacturer to address the question of software management and QA by taking into account the specific problems of the trainer industry and the unique management objectives within the company. The existing military standards and specifications provide a model which can be analyzed and tailored to meet the needs of a company program. Perhaps, once software control has been addressed on a company by company basis, a set of software standards applicable to the trainer industry, coordinated and approved by all industry representatives, might be written. This would provide the final solution to most of the questions addressed by this paper.




SOFTWARE DEVELOPMENT SCHEDULE

FIGURE 1

## BIBLIOGRAPHY


Boehm, B. W., "Software and Its Impact: A Quantitative Assessment," Datamation, Volume 19, No. 5, May 1973.

Buckley, Fletcher, "A Standard for Software Quality Assurance Plans," Computer, August 1979.

McKissick, John, Jr. and Price, Robert A., "Quality Control of Computer Software," ASQC Technical Conference Transactions, 1977.

Mendis, Kenneth S., "A Software Quality Assurance Program for the 80's," ASQC Technical Conference Transactions, 1980.

Prudhomme, Robert R., "Software Verification and Validation and SQA," ASQC Technical Conference Transactions, 1980.

Thayer, T. A., "The Role of Product Assurance in Improving System Reliability," TRW Systems, Redondo Beach, California.

Thayer, T.A., "Software Reliability Study Final Technical Report," prepared for Rome Air Development Center, March 1976.

Walker, Michael G., "A Theory for Software Reliability," Datamation, September 1978.