SOFTWARE QUALITY ASSURANCE (SQA) EARLY IN THE ACQUISITION PROCESS

Ronald A. Simmons

Modern Technologies Corporation
Dayton, Ohio

ABSTRACT

Experience is showing that the earlier Software Quality Assurance (SQA) is involved in the acquisition phase, the more effective it becomes. However, SQA is assigned relatively late in the development phase of a weapons system program becuase the software coding does not occur until late in the development phase (deliverable software). This is a result of software quality assurance programs having been generally administrative rather than technical.

SQA, though, is a rapidly evolving discipline, with more and more attention being given to quantitative measures of quality. Due to the software constraints being applied, e.g., reliability, maintainability, interoperability, usability, etc., quantitaitve quality factors have been developed to be applied agaionst the software design, the developed code, and the delivered code to measure the software's capability to meet these constraints. In order for these criteria to be met, they must be stated up front, designed in, and effectively measured.

The purpose of this paper is to show that there are certain milestones in the software acquisition process where SQA activities have the greatest effect on the software acquisition process. During certain phases of this acquisition process, certain quality assurance activities are necessary in order to provide the greatest effect on software quality. Conversely, after certain milestones, it will be shown that it is no longer feasible or cost effective to apply specific SQA actions. This paper will show that SQA processes can be applied in each phase of the software acquisition process to maximize quality.

INTRODUCTION

Software is becoming more complex, more critical to successful operation of a weapon system, and more costly to develop. Errors found in the software during development are also becoming more costly to correct, and can cause significant schedule delays in fielding a weapon system or, if not corrected, cause degraded operational capability. This paper will show some of the software trouble spots and some effective software quality assurance (SQA) techniques and tools that can be applied to reduce or mitigate the risk. Software errors vary in number and magnitude depending on the time phase of the development cycle. Thus, the effectiveness of the SQA tools and techniques to be discussed is dependent on the acquisition phase to which they should be applied. Figure 1 depicts the increasing dependence on embedded software of some major Aeronautical Systems Division (ASD) weapons systems. The amount of software being used in major systems has increased exponentially over the last two decades. What this chart does not show, but what can be implied, is the criticality of that software to successful performance of the mission.

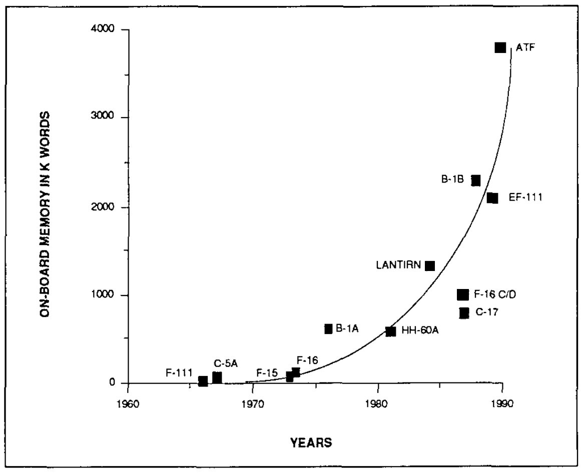

Figure 1 ASD Weapon Systems Embedded Software

Each of the systems shown in Figure 1 has had significant software problems that have caused schedule delays, cost overruns, and degraded operational performance; some systems still have software problems. This is becoming increasingly typical for major systems as more reliance is placed on software. Some of the problems are inherent in the DOD acquisition process; some of the problems are due to systems having to be fielded before they have been fully tested; some are due to poor design and many are due to changing requirements during the development process.

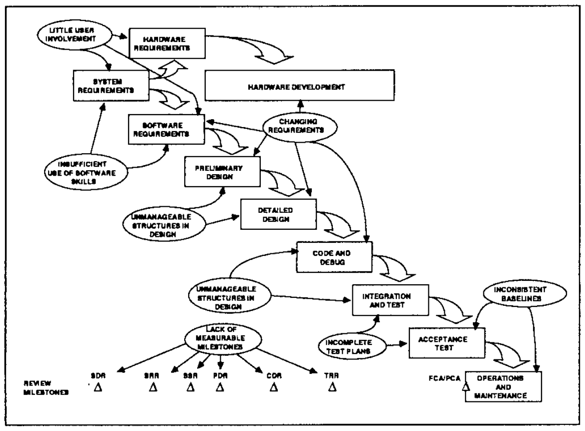

Figure 2 Typical Software Trouble Spots

ANATOMY OF THE PROBLEM

Figure 2 shows the software acquisition cycle and the typical software trouble spots that have occurred on most major software acquisitions. These software trouble spots cause latent defects in the software that are not usually found until late in the software development. Little user involvement in the early stages of program and requirements definition affects supportability as well as operational suitability. As the user becomes more involved, requirements change to more closely meet the user's need. Requirements analysis and requirements allocation to configuration items are made with insufficient thought given to whether software can best meet the requirement, resulting in logic designs that are unwieldy. The acquiring agency also has few review points or measurable milestones during the development process. Incomplete test plans result in software that has not been fully tested, resulting in stretching out the test phase or accepting software that may not work as required. Often the software that undergoes acceptance is not the same software that ends up in the field. While these are the typical software trouble spots, they only serve to hide the real problem: late identification.

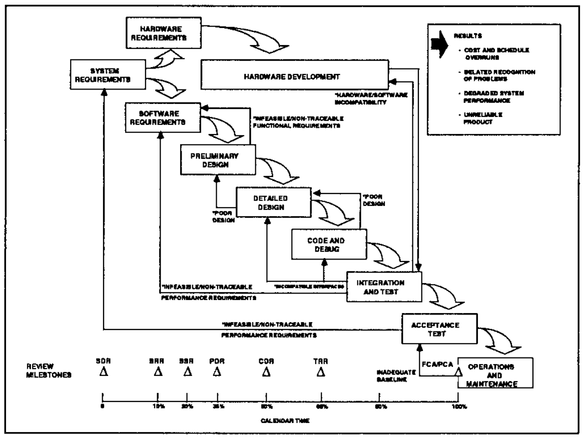

Figure 3 Anatomy of "Real" Problem: Late Identification

Figure 3 depicts the relationship between time of identification of the problem and in what stage of development the defect actually occurred. What this shows is that generally the most significant or difficult problems occur early in the development cycle but are not found until integration and test have begun. This is not to say that errors are not found earlier in the development cycle. Errors found early in the design phase can be attributed to infeasible or nontraceable functional requirements and poor design. Errors found during code, debug and unit test are attributed to poor design. However, when the software begins the integration and test phase, incompatibility of interfaces between CSCIs begins to show up, as well as the hardware-software incompatibilities. These can then be traced back to the system requirements phase as infeasible and nontraceable performance requirements, indicating improper allocation of system requirements and unrealistic or unrecognized performance requirements. Looking at the bottom of Figure 3, you can see that almost all of the major review points in the development phase occur before the development is 50% complete, with almost no review milestones occurring until program completion. Yet most of the major problems are found late in the development. Corrective action that must be taken to correct the errors found in the late stages of development usually takes the form of Engineering Change Proposals (ECPs) that increase cost and cause schedule delays.

If these problems can be found earlier, preferably in the stage of development in which they occur, the cost to correct is small and will not cause significant schedule delays. Finding and correcting problems early will also result in better system performance and a more reliable product.

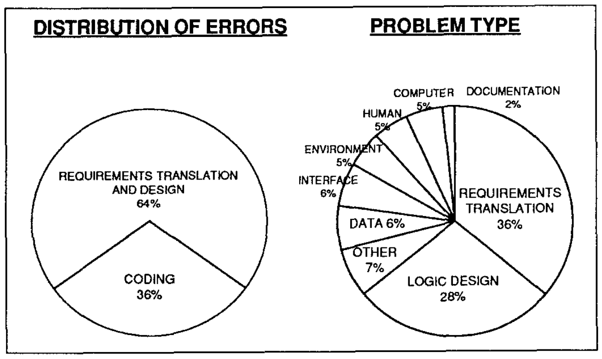

Figure 4 ASD Embedded Systems: Software Problem Type and Time of Occurrence

Figure 4 shows the types of problems found and the distribution of errors for some major ASD programs. Thirty-six percent of the problems are directly attributable to poor or inadequate requirements translation, which occurs early in the development process; yet these types of problems do not surface until integration and test begins. By this time the design is fixed and the problem is very costly to correct. Twenty-eight percent of the problems are due to logic design flaws that also occur early, but are not found until the program is into detailed design and code phases. Fully 64% of the errors found are the result of problems that occurred early in the development process, yet were not found until late in the development. Only 36% of the errors are due to coding flaws. With classic SQA under MIL-S-52779A, the coding errors are the only errors that could be found and corrected in a timely fashion. With the advent of DOD-STD-2167, DOD-STD-2167A, and DOD-STD-2168, the potential to find the other 64% of the errors and correct them in a timely fashion can be realized.

It is obvious that correcting an error as soon as it occurs is cost effective. However, it is not so obvious what the cost is to correct an error later in the program. Fifty-four percent of all design and coding errors in a software development program are discovered late.

• 54% OF DESIGN & CODING ERRORS DISCOVERED LATE

DISCOVERY PHASE	INCREASED COST
CODING/BUILD	1.5 TO 3
INTEGRATION	2 TO 5
VALIDATION	3 TO 10
FLIGHT TEST/OPERATIONS	10 TO 90

Figure 5 Cost of Late Discovery and Correction

Figure 5 shows how the cost to correct increases drastically the later the error is found. Requirements related problems found and corrected during flight test can cost over 90 times what it would have cost to correct during the systems requirements phase. It is for this reason that programs are cancelled, incur cost and schedule overruns, or suffer degraded system performance when fielded.

The question that must be answered then is, can these errors be found early enough to be corrected without major impact to the program? Let us now examine that issue.

TIME DEPENDENT ISSUES

First, we need to know when the impact of SQA or lack thereof is first and most apparent, then work backwards to determine when it should be planned and applied in order to be most effective. This is rather like having the answer, then trying to find out the question, but for our purposes will serve to show just how early SQA needs to become involved in the program and acquisition planning. This method also illustrates that there is a "point of no return" from which it is no longer feasible or cost effective to implement SQA activities for certain tasks.

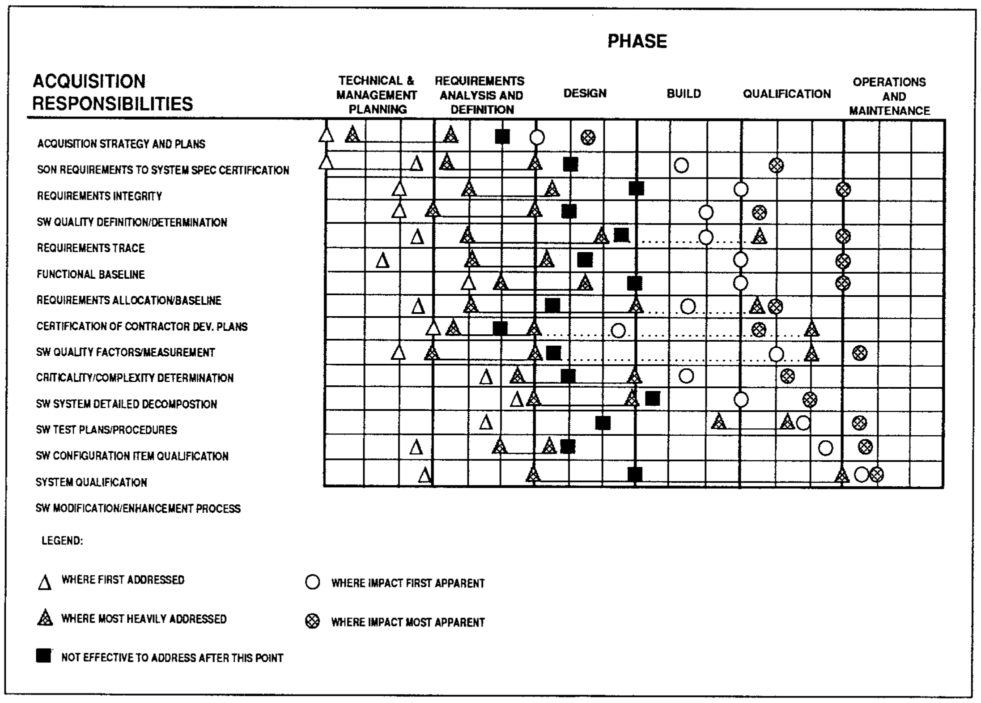

Figure 6 Acquisition Events Versus Phase

Figure 6 shows the system acquisition process broken out into six phases: technical and management planning, requirements analysis and definition, design, build, qualification and, last, operations and maintenance. The column labeled acquisition responsibilities lists certain tasks that involve SQA activities necessary to the successful completion of the program.

The impact of having specified quality assurance goals, objectives, needs and plans for achieving them in the acquisition strategy and plans first becomes apparent at the beginning of the design phase and is most felt during the design phase. For hardware, the same is true for such things as producibility, supportability, maintainability, and reliability. If those are not addressed in the acquisition plans and the SOW, they cannot be designed in later without great additional cost and schedule overruns.

As for hardware, so it is for software. Software quality factors such as reliability, maintainability, portability, expandability, and interoperability must be planned in if the design is to reflect those factors. It is axiomatic that quality must be planned and designed into the software. It cannot be added later. If the QA goals and needs are not addressed during the requirements analysis phase, the software designer cannot include them in his design; it is too late to cost effectively add them after this point.

Thus attention must be given very early in the acquisition phase to software quality requirements and plans in order for them to have any impact on the design. This planning process for software quality will have a ripple effect throughout the program life cycle. As you can see in Figure 6, SQA has an effect even in the operations and maintenance phase of the program.

If you study Figure 6 carefully, you will note that software quality must first be addressed in the planning stages and in the requirements analysis phase, and, if not addressed then, becomes cost ineffective during the design phase. However, you will also note that the impact of software quality does not readily become apparent until the build (or code and unit test) and qualification (or integration and test) phases.

An example of the impact of SQA planning not becoming evident until late is requirements traceability. Requirements traceability, both upwards and downwards, must be specified in the SOW and the requisite documentation contracted for, the methodology and tools and verification techniques spelled out in the Software Development Plan (SDP) and in the Software Quality Program Plan (SQPP). The worth and utility of requiring traceability and verifying it do not become apparent until you are almost ready to start testing. It becomes a most valuable tool during the test phase where it can be used to easily verify and validate that all the requirements allocated to software have been met. Of course, it can be used much earlier than during qualification testing. It is also of considerable value (during the Functional/Physical Configuration Audit (FCA/PCA) prior to turnover) to the operational user to ensure that the software meets not only the contractual requirements, but also the user requirements. If levied early enough, this tool allows SQA to find and point out deficiencies and start corrective action before they become major or costly problems.

Automating the requirements traceability can also make the developer's and the acquisition agency's job easier when a change in requirements occurs, as it will readily indicate the impact of the change and give a relative magnitude of the change under consideration.

However, if there is no requirements traceability required of the developer, or SQA is not involved in this process, there comes the "point of no return" late in the design phase where the cost to correct a design flaw or missed requirement becomes too expensive in both cost and schedule. This then results in degraded operational capability or, if severe enough, can result in cancellation or restructuring of the entire program.

TECHNIQUES AND METHODS

What the SQA engineer needs to become is the early warning system for the software engineer. Software problems are generally difficult to solve because they surface without warning, long after the problem has occurred. Timely detection and corrective action are the key to producing quality software that meets the requirements.

The methodologies shown in Figure 7 are simple but need careful thought in planning and execution. As you can see from the matrix, the techniques and methods are used to some extent throughout the development process. First you need to determine to what extent and how these methods will be utilized for the development effort, then establish SQA policies and standards that are tailored to the particular development effort. The quality assurance goals or scope of effort must be determined, and the SQA tools and techniques selected for each phase of the program. Then the SQA plans need to be written and tailored to the development effort. This all should occur in the technical and management planning phase of the program.

	TECHNIQUES/METHODS										
SOFTWARE QUALITY ASSURANCE ASPECTS	PAERTO ANALYSIS	REQUIREMENTS TRACE	SW MATURITY DETERMINATION	SIMILAR RECURRING PROBLEM IDENTIFICATION	RISK CONTAINMENT ANALYSIS	DEVELOPMENT PLAN ASSESSMENT	PARSING TECHNIQUE	QUALITY FACTORS/ MEASUREMENT	VERIFICATION LEVEL AND METHODS ASSESSMENT	DEV. AND LIBRARY CONTROLS ASSESSMENT	STANDARDS & PROCEDURES CORRESPONDENCE ASSESSMENT
TECHNICAL AND MANAGEMENT PLANNING	X	X	X	X	X	X	X	X	X	X	X
REQUIREMENTS FIDELITY (COMPLETE, CONSISTENT, UNAMBIGUOUS, MEASUREABLE)	X					X	X	X	X		
REQUIREMENTS ALLOCATION	X	X		X	X	X		X	X	X	X
REQUIREMENTS CRITICALITY	X		X	X	X	X	X		X		
RISK IDENTIFICATION	X	X	X	X	X	X			X		
SYSTEM/SOFTWARE DECOMPOSITION	X	X	X	X	X	X		X		X	X
REVIEWS (DEVELOPMENT, DESIGN, BUILD, TEST READINESS)	X	X	X	X	X	X		X	X	X	X
SOFTWARE SYSTEMS/PRODUCTS COMPLEXITY	X			X	X	X	X	X	X		
SOFTWARE BASELINES		X			X	X		X		X	X
DT&E AND PRODUCT QUALIFICATION (PQT, FQT, SYSTEM QUAL.)	X	X	X	X	X	X		X	X	X	X
AUDITS (FCA, PCA, FQR)	X	X	X	X	X	X		X		X	X
MODIFICATIONS/ENHANCEMENTS	X	X	X		X	X		X	X	X	X

Figure 7 How It Is Applied

During the requirements analysis and definition phase the development risk must be analyzed and determination made of where the highest risks are; then an analysis must be made as to how to contain or mitigate those risks. Part of the planning should be what quantitative measures can be made during the development process to assess the risk and provide feedback to management.

Careful selection of software quality factors for each phase of the process will provide the assessment criteria for later measurement. The framework set up at this time will determine the success or failure of the SQA efforts. If little attention is paid at this point to SQA, the payoff during integration and test will not be very great. Careful attention to detail and some additional effort spent in requirements analysis, requirements translation, and requirements decomposition by SQA will result in significant cost savings later. In fact, the investment in SQA early in the development process can provide cost savings and prevent schedule delays later in the program.

It is at this time that requirements

traceability must be established, although traceability will continue until turnover to the maintaining command. If requirements traceability is the function of software engineering, it is all the more important that SQA verify the traceability. As part of the verification of requirements traceability, SQA should also verify that the requirements allocation to the individual configuration items is complete, functionally grouped and reasonable.

Another technique that needs to be planned and implemented early in the acquisition process is Pareto analysis, which should be conducted on a continuing basis until transition to the maintaining command.

Other techniques to be planned and implemented at this stage include development plans assessments, software development library and configuration control assessments, and continuing assessments of standards and procedures.

LESSONS LEARNED

More and more attention is being focused on software quality assurance, and more investment is being made in SQA. SQA is truly an evolving discipline, and we are learning many valuable lessons from our mistakes.

A 1983 survey of 135 organizations showed that 108 organizations had less than 1% SQA, and only 3 had more than 4% SQA. Yet a 1986 GAO survey of successful government software projects showed that the SQA staff averaged 6.9% of the software development effort. This is a dramatic increase in staff and cost, but the investment obviously paid off.

Some companies today have a much higher investment in SQA, with some in the 12% to 16% area. Even though these companies have a larger SQA staff, they have not yet made the investment in automated tools that they have in software development tools, although, admittedly, automated SQA tools are still in their infancy.

SQA is frequently not budgeted for, not planned, or planned late in the development process, and is viewed as an administrative function. SQA is phased in too late to be effective, therefore has no influence on building in quality. The result is a belated attempt to add on quality at best. There are generally no specific SQA requirements in the RFP. Either there is no statement of how software quality will be measured, or this statement is general or vague and the compliance regulations are too general to allow enforcement.

SQA is generally not involved in requirements traceability issues, and software engineers do not use requirements traceability tools.

As already mentioned, SQA tools are generally lacking, not equal to the software development tools, or unable to interface to them.

While there may be great attention to software trouble reports, little attention is paid to tracking the corrective actions. Trend analysis is rarely accomplished and, if the analysis is done, the results of the trend analysis are not interpreted or presented to management in a manner that would allow corrective action.

CONCLUSIONS

Embedded software development has a long history of development problems. These problems usually occur very early in the development but are not found until late in the development. This results in degraded operational capability, or cost and schedule overruns, or both.

It is indeed axiomatic in the software world that you *cannot* add quality; it must be designed and built into the product. In order to obtain quality software, you must plan for it carefully, design it in, then build it in. In order for SQA to be effective, it must be planned very early in the acquisition process, usually during the demonstration validation phase just prior to the start of full scale development. Software quality requirements must be part of the proposal process and part of the system specification.

An aggressive SQA approach is required in order to find and correct problems when they occur. SQA is another one of those areas that the program manager must work "up front-early on" if he has any expectation that his program will be successful, instead of being cancelled.