

A REVIEW OF COMPUTER CONTROLLED SYSTEMS SAFETY AND QUALITY ASSURANCE CONCERNS FOR ACQUISITION MANAGERS

by

John Robert Friend
Commander, United States Navy
Polaris Missile Facility
Charleston, S. C. 29408

ABSTRACT

The reliability and safety of computer controlled devices is becoming more of a concern when these devices are used to control real time, critical systems. Often the acquisition manager or purchasing agent for a computer controlled system has no knowledge of software design criteria. This paper reviews the basics of software engineering, applicable safety and quality assurance standards and how to employ these standards. Included is a discussion of structured programming, verification and validation of completed program code and testing. As a part of this discussion the software life cycle phases are highlighted with a description of how each commercial and government standard relates to a particular phase.

The difference between software safety and quality assurance is a major focus. The report describes the Department of Defense safety standard and details the tasks within the standard which relate to software safety. Specific recommendations for changes in Department of the Navy requirements for the use of software standards are outlined.

Introduction.

I first became acquainted with the term software quality assurance when constructing a computer simulation of a guidance system for a masters thesis. In an effort to streamline the code, I found that reversing two of the graphics calls caused the program to "bomb". After many hours of review and some assistance from a friendly professor, the cause of the difficulty was discovered to be a "well known" error in the operating system of the work station being used for the project. The thought that an established operating system could have such a flaw was really a surprise. This experience came back to me when the weapons facility where I am presently assigned procured a new computer controlled crane which experienced a few computer related problems as it was being placed in service. In this case software quality assurance could easily become a safety concern. The issue of safety is particularly significant considering the sensitive material that is handled by this type of crane.

Safety especially software safety must be built into such a system in the earliest design stage. These safety concerns and how these concerns are included in commercial, Department of the Navy and Department of Defense

Acquisition Procedures are the basis for this paper.

Purpose.

The purpose of this report is to provide the reader with a discussion of software safety and software quality assurance as it refers to the Department of Defense (DOD) and commercial material acquisition process. Quality assurance and safety are interrelated and require a systems approach to design of both hardware and software. This paper is divided into three sections:

- 1) A review of software safety and quality assurance and a discussion of methods presently used to implement them.
- 2) A discussion of commercial and Department of the Navy (DON)/DOD standards for implementing software quality assurance and safety during the acquisition process.
- 3) The final section includes guideline for the acquisition manager on what areas should be reviewed to provide a chance of getting an acceptable product. Additional recommendations are provided for users who are already in possession of a computer controlled product and are unsure whether the proper reviews were completed during its procurement.

Assurance versus Safety

Reliability and safety are often considered synonymous, particularly when these concepts are applied to computer software [1]. In fact, they are different. Reliability is the ability of the system to remain failure-free while safety is based on the ability of the system to remain hazard-free. The word system is used because both reliability and safety must be measured as function of the hardware and

software involved. The software may be perfectly designed but if the interface between hardware and software is improper or external factors such as the human interface are not considered, the system may malfunction. Increasingly, computers are being used to replace classical control systems because of the complex nature of the processes involved, the ability to reduce the weight and cost of some systems and the belief that the computer is inherently safer than its analog counterparts. This last belief is not necessarily accurate.

Accidents involving computer controlled devices frequently occur. Leveson [1] in her paper on software safety describes an incident where programmers were designing software to control a chemical reactor. The design engineers had instructed the programmers to leave all controlled variables constant and sound an alarm if a fault occurred. An alarm was received telling the computer that there was a low oil level in a gear box. The alarm proved to be false but because a catalyst had just been added, the amount of heat produced by the reaction was rapidly increasing. The computer froze all controls constant including the control of the water flow through the condenser. This caused the reactor to overheat, the relief valve to lift and the entire contents of the reactor was vented to the atmosphere. The error here was that there had been no systems review of goals for the project nor a systems safety plan to avoid potential hazards.

Safety-critical systems typically have reliabilities from 10^{-5} to 10^{-9} [1]. Several studies have indicated that computer controlled systems may be several orders of magnitudes worse than this. Software adds multiple layers of complexity. Often reliability suffers because the computer has more features than what is actually needed for the task which is being performed. These features may cause additional

failure modes. Safety must be designed into the system from program inception. It is important that a safety plan be developed which includes a definition of safety requirements, hazard states and testing procedures [2] [3]. As much as practical, during the development process, the designer must attempt to define all system states, determine which states must never occur and then define those states which would directly lead to hazardous situations. This process is diagrammed in Figure 1. System design must then be modified to minimize the hazardous conditions. It is important to identify those states which are hazardous and develop software design requirements to minimize their impact.

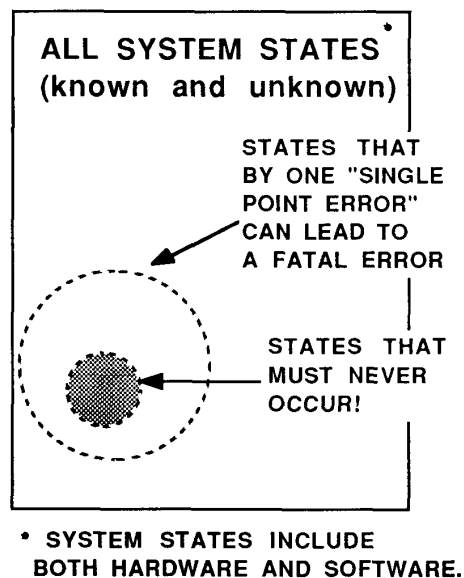


FIGURE 1.

Several techniques have been used for this type of analysis [1][4]. One of these is Fault Tree Analysis (FTA). It has been used in the past for electromechanical system safety reviews. In this technique an undesirable end state is identified and then the system is examined to determine probable parallel and serial sequences of events which could cause the unsafe condition. The fault tree can be represented graphically. The events leading to a fault may include hardware, software and human interface considerations. Probabilities could be assigned to determine the likelihood of each critical event occurring and a probability for the undesirable end result determined. The advantage of this method is that all system components can be considered in the analysis. The disadvantage is that the quality of the product is highly dependant on the proficiency of the individual conducting the analysis and their familiarity with the whole system [1].

Figure 2 shows an example of a FTA for a potential hazard for a large wharf crane used in moving heavy cargo off of naval ships. The hazard that is defined is structural failure of the crane after the crane stops with a large load suspended in a strong wind. The identification of a list of potential hazards should be done as a part of a preliminary hazard analysis done during the initial design phase. A separate tree is created for each hazard. The tree starts with the hazard as its root with branches working back to the initial conditions which would cause the fault. The types of event sequences which must be examined as part of

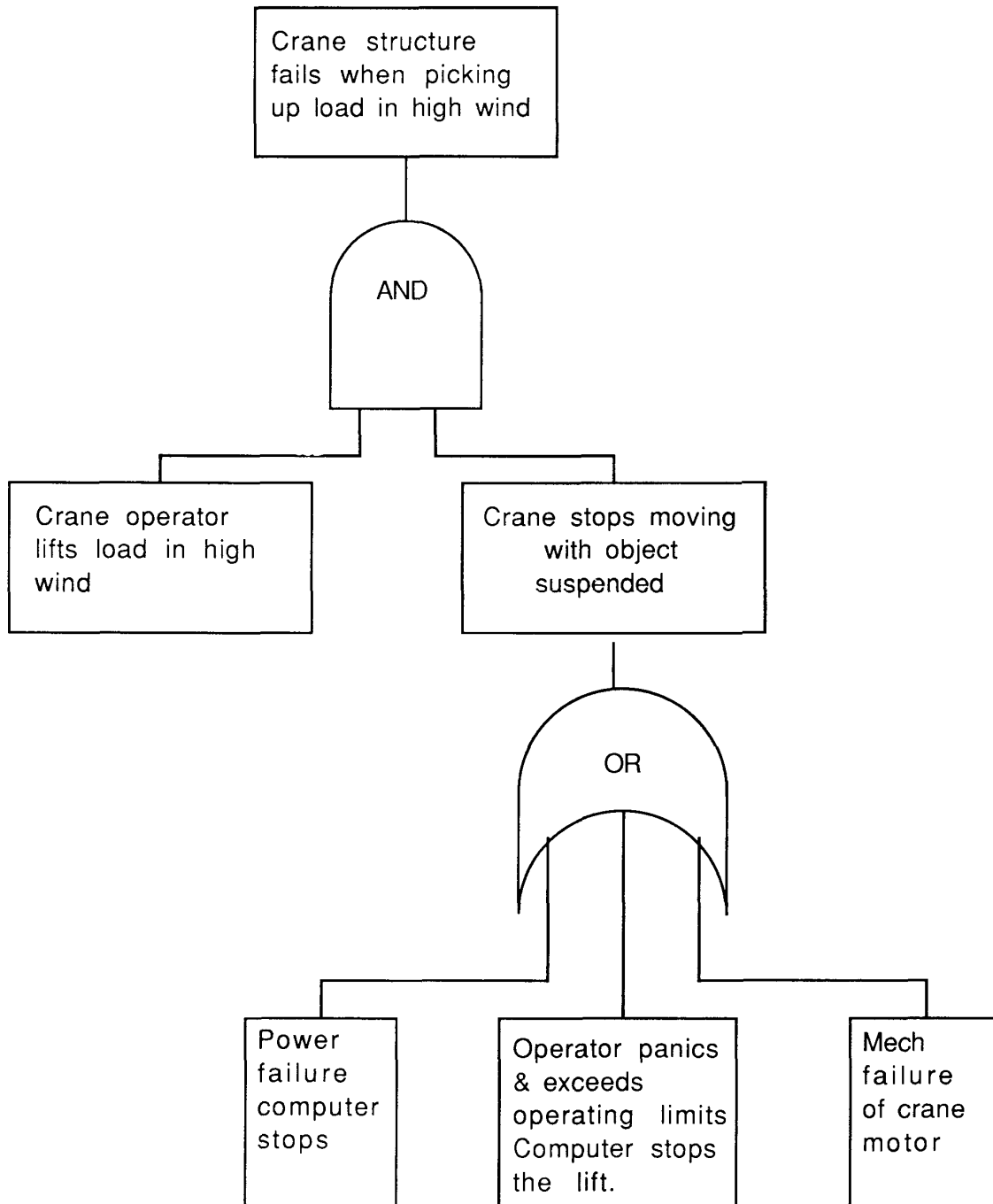


FIGURE 2.

generating such a tree include failure to perform a required function, performing a function not required, timing problems and incorrect response to hazardous conditions [1]. In the case of figure 2 two separate events must occur. The operator must pick up the large load during excessively windy conditions and the crane hoist motor must fail. The combination of both events causes an over stress condition on the crane for lengthy period of time. The result is structural failure. Several separate events can cause the hoist failure and this is represented by the OR symbol which connects these events in figure 2 while the AND symbol is used to show eventualities which depend on a combination of several other events to happen. Other methods of analysis which been developed are Nuclear Software Cross Check Analysis (NSCCA) developed for the U. S. Air Force and Software Common Mode Analysis [1]. NSCCA is a two step process which is made up of the identification of specific requirements to ensure the minimum measures for nuclear weapon safety have been met and a review of each software operation to determine whether it influences a critical nuclear weapons event. A qualitative assessment is made to determine the degree of influence on each event and and this assessment is used in the development of the system test plan.

Software Common Mode Analysis is used to evaluate the redundant features used in critical systems. A hardware failure the affects multiple redundant elements is called a common mode failure. In this method all hardware software interface points are identified. The ability of one hardware fault to propagate through the software to other systems is determined.

To analyze software for safety and reliability a well defined program

structure is required to assist in software review and design. Safety and reliability are implemented using a structured design process.

When discussing structured software design, the terms validation and verification are often mentioned. Simple definitions of verification and validation are listed below:

"Verification is an attempt to find errors by executing a program in a test or simulated environment".

"Validation is an attempt to find errors by executing a program in a given real environment". [4]

However, the sense of what the terms verification and validation (V&V) symbolize can be broadened by some authors from these simple definitions to more comprehensive definitions which encompasses the whole structured programming management process. Dunn's definitions [5] of V & V expands the definition of verification. He stated it as "procedures that attempt to determine that the product of each phase of the development process as an implementation of a previous phase; that is, that it satisfies it. Thus, the design of the components of a system are examined to see if they meet the external characteristics specified for them in the top-level design." He defines validation as "Procedures that attempt to determine that the product of each phase of the development process will lead to satisfaction of the most abstract requirements set."

What is being described here is a management technique which uses frequent varied reviews and testing to determine conformance to specification and which relies on structured programming techniques to provide auditable software which can be easily modified and maintained. Fuji and Wallace in National Institutes of

Standards Publication 500-165 [6] define the objectives of software V & V as a process to comprehensively analyze software during development and maintenance to:

- 1) determine correct operation
- 2) ensure that it performs no unintended functions
- 3) measure quality and reliability

They go on to break down V&V into phases with specific tasks and key issues. These phases are defined as concept, requirements, definition, design, implementation, test, installation and checkout and operation and maintenance.

The concept, requirements and definition phases involve in-depth discussions between the potential users and the management team which is constructing the system. These phases are critical both for ensuring the system can fulfill its intended mission by formulating the correct requirements which state what the system must do and by conducting the initial in-depth safety survey which will tell the designers what the system must not do.

Top down design and structured programming techniques are heavily used in the design and implementation phases. The design philosophy starts with a list of specifications which are used to formulate the flow of control for the program. Initially, the flow is represented by a few complex modules. As the design progresses, these modules are broken down into smaller modules with simple defined purposes. These simpler modules are easier to design and understand. In general, the programming code can be written in a simple and uncomplicated manner. "One important feature of top down design is that at each level the details of the design at lower levels are hidden. Only the necessary data and control which must be passed back and forth over the interface are defined [5]".

Figure 3 shows a basic flow diagram example of top down programming where a central main program calls on individual modules to perform simple specific tasks. Figure 4 shows the same program written in a pseudo C language code. The computations shown would be performed in separate callable routines in a more complex example. The top down approach combines well with basic structured programming which relies on a single decision module and one main program loop.

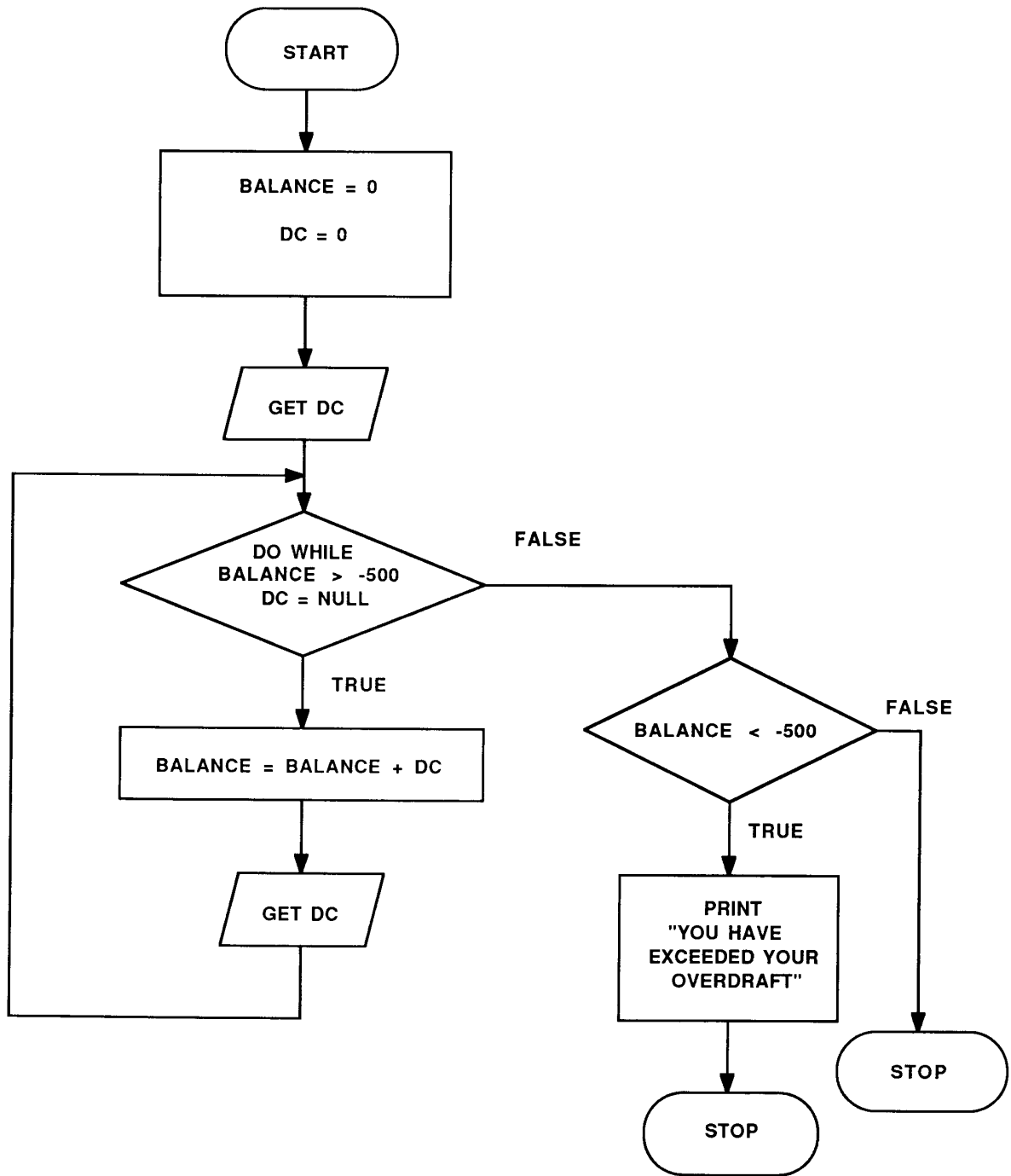


FIGURE 3.

```

/* This program is a simple
   checking program */

main
{
    DC = READ(INPUT LIST);

    WHILE BALANCE > -500
    AND DC ≠ NULL

        {
            BALANCE = BALANCE + DC;
        }

    ELSE IF BALANCE > -500

        {
            STOP;
        }

    ELSE

        {
            PRINT " YOU HAVE
                  EXCEEDED YOUR
                  OVERDRAFT" ;

            STOP;
        }
}

```

FIGURE 4

In structured programming modules addressed should have one entry/exit point [4]. The "if then else" structure in the C and Ada programming languages are specifically designed for this method of programming. "If then else" constructs are preferable to the "go to" structures used by older programs because the "go to" structure is much more likely to produce an error if the program is interrupted during the "go to" jump. The module approach facilitates easy

programming code maintenance, enhancements and makes code reviews of the software more straight forward.

Ada has been mandated by the DOD as the computer language of choice for all major computer programming projects. It was chosen because of Ada's structure and unique error handling capabilities. Its built-in error handling capability, often referred to as exception handling, allows programs written in this language to detect errors caused by defective input devices or out of parameter inputs which disrupt program flow and compensate for them. Ada is a relatively new computer language which is complex. Because of this complexity the number of programmers proficient in its use is small. This fact tends to drive up cost when Ada is used. However its error handling capabilities make it ideal for automatic control systems and outweigh the cost penalty, especially for critical systems.

The interface between the operator and the system and the human factors which must be considered is another critical element in the design process. The decision concerning how to divide tasks between man and computer is fundamental to the design process. Tasks can be partitioned between the operator and computer or tasks can be allocated based on which has the most resources at the moment [1]. An example of shared decision making is an air traffic control system where the computer provides alternate routes for aircraft and warns of potential conflict. The controller can use the recommended computer routing or choose alternates depending on the time available and his knowledge of other factors such as weather that the computer may not have taken into account of when the preferred route was provided.

Other human interface considerations are operator complacency, the amount and type of information presented to the operator, and the confidence the user has in the system as a whole. Complacency can develop when a complex process is entirely directed by the computer except when an emergency occurs. Then the operator is thrown into a difficult situation where his skills may not be sufficient to succeed because of his lack of familiarity with the control system. A decision of this type developed early on in the nuclear submarine program where an automatic depth keeping system was designed to maintain ship depth with the operator intervening only when high sea state or high speed operations forced shifting to manual control. It was found that the sailors who were assigned to operate the depth keeping planes did not maintain sufficient skill to succeed in more difficult situations unless they were the primary control during more routine operations. As a result many commanding officers chose to operate their systems in the manual mode.

Operator confidence in a device can be shaken when the information presented by the system is unfamiliar and confusing or if the machine continually stops because of over sensitive control circuits. A lack of operator confidence can cause avoidance of the machine or worse bypassing of safety features to avoid stoppages. In designing the human interface it is important to remember that both humans and computers make mistakes and the goal should be to design an interface which optimizes the performance of both.

Testing and final checkout are the remaining development phases. The primary purpose of the test plan is to maximize the number of errors found while minimizing the amount of testing.

It is impossible to find 100% of the errors in any program exceeding a few hundred lines in length. The best that can be done is to find as many as possible. Testing is in reality done throughout the design and implementation process. Code reviews are used to verify correctness, adherence to standards, and as a check on programming logic. These reviews may be done by one individual or by an organized group. Ideally, the review should be totally independent of the original design and programming organizations. A well defined program structure and well commented code are important to ensure traceability by the reviewers. Code reviews or walk throughs can make up a large proportion of the programming effort in complex safety related programs. In some of the space shuttle software, reviews accounted for 75 percent of the programming effort and raised the cost to six hundred dollars per line of code [5]. Some informal cost estimates for the TRIDENT Submarine Navigation Program are as high as 2400 dollars per line. Static reviews involve code reviews by machine and are heavily used to detect unused code fragments, improperly defined constructs and error trapping of defects not caught by manual reviews.

Dynamic testing of individual modules and the fully integrated program is also conducted. Two terms that are often used are black box and glass box testing. Black box testing refers to testing a program or module with respect to its external specifications. When glass box testing is performed, the testers know everything about the program. These tests are designed to test specific program flow paths [5]. Acceptance testing is in general black box testing.

To summarize this section, the complexity of the definition of software verification and validation varies with the

author. In general verification measures how well the software performs against the stated requirements while validation measures performance in the actual operating environment. Structured programming is a useful tool used to make certain that stages of the V&V process (Concept, Definition, Design, Implementation, Test, Installation and Checkout) can be effectively implemented. Safety must be built into the system in the concept and design phases. V&V is important to software quality assurance and software safety. However quality assurance is the attempt to verify that the program performs all required tasks while software system safety is the effort to assure that the systems has only a small probability of operating in an unsafe manner.

Software Safety and Quality Assurance Standards

The purpose of standards for both safety and quality assurance is to define the basic requirements for project management and documentation for the life cycle of the product. Table 1 lists software safety and quality assurance standards which provide guidance on how best to establish the best management structure for a particular project. The annotated publications in Table 1 [3, 6-20] specifically address V&V. All the listed standards provide direction to determine how well the software products comply with stated requirements and guidance on how to select techniques to satisfy application needs. The guidance is broken down into discussions of organizational structure, management, documentation, management reviews

and audits, software configuration management, life cycle maintenance and software test management. These items are normally a part of an overall Software Quality Assurance Plan (SQAP).

Organizational concerns center on the management structures which influence quality and principally whether the V&V activities are performed as a part of the developers normal quality assurance sections or by an independent organization. If subcontractors are used for development, a procedure for examining their work in terms of an overall plan must be included. Figure 5 [21] diagrams the hierarchy of a complex process where the end user has commissioned an independent V&V team. The developer has his own team which defines requirement for the V&V effort at each of the subcontractors.

This type of project is typical of government weapons contract. DOD-STD-2167A requires that the developer have a quality assurance section, but individual armed services and internal service organizations require independent V&V, an example of this can be seen in Air Force pamphlet AFSC/AFLCP 800-5. Federal Information Processing Standards Publication (FIPSPUB) 101, American National Standards Institute (ANSI)/Institute for Electrical and Electronic Engineers (IEEE) standard 983, ANSI/IEEE standard 1012 and ANSI/IEEE standard 730 provide guidance for software produced for commercial uses. These publications do not require an independent V&V organization but require that the relationship between the V&V group and other quality assurance functions be strictly defined. This guidance is similar to that provided to the nuclear industry as described in American Nuclear Society (ANS) Standard 10.4

STANDARD NUMBER	NAME	V & V DISCUSSED
FIPSPUB 101	Guideline for Life Cycle Validation Verification & Testing of Computer Software	YES
FIPSPUB 132	Guideline for Software Verification and Validation Plans	YES
ANSI/IEEE 610.12	Glossary of Software Engineering Terminology	YES
ANSI/IEEE 730	Software Quality Assurance Plans	YES
ANSI/IEEE 828	Standard for Software Configuration Management Plans	
ANSI/IEEE 829	Standard for Software Test Documentation	
ANSI/IEEE 830	Guide for Software Requirements Specs	
ANSI/IEEE 983	Guide for Software Q A Planning	YES
ANSI/IEEE 1012	Standard for Software V & V Plans	YES
ANSI/IEEE 1016	Recommended Practice for Software Design Descriptions	
ANSI/IEEE 1028	Standard for Reviews and Audits	
ANSI/IEEE 1042	Guide to Software Configuration Management	
AFSC/AFLCP 800-5	Software Independent V & V	YES
ANS 10.4	Guidelines for the V & V of Scientific and Engineering Computer Programs for the Nuclear Industry	YES
DOD 2167A	Military Standard Defense System Software	YES
DOD 2168	Mil Standard Defense System Software Quality	
MIL 882B	System Safety Program Requirements	

Table 1.

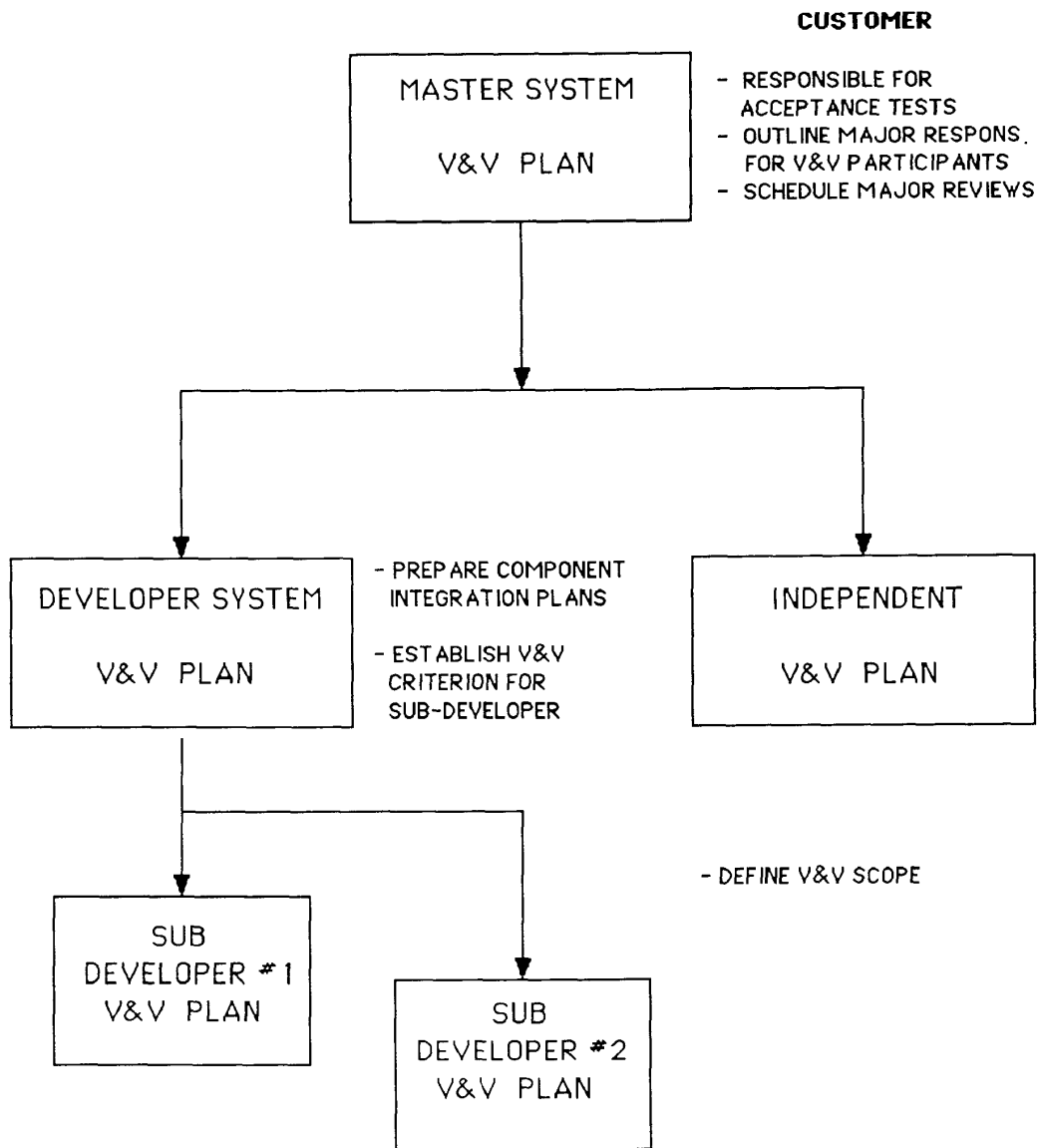


FIGURE 5.

TITLE	DESCRIPTION
Software Requirements Specification (SRS)	"Documentation of the essential requirements of the software and its external interfaces"
Software Design Description (SDD)	"A representation of a software system created to facilitate analysis, planning, implementation, and decision making. A blueprint or model of the software system. The SDD is used as the primary medium for communicating software design information."
Software Verification and Validation Plan (SVVP)	"A plan for the conduct of software verification and validation."
Software Verification and Validation Report (SVVPR)	"Documentation of V&V results and appropriate software quality assurance results."
Software Development Plan (SDP)	"A project plan for a software development project."

Table 2.

Documentation requirements should cover all directives governing development, verification, validation and maintenance of the software. A discussion of how these documents will be checked for accuracy must be included. As a minimum, a Software Requirements Specification (SRS), a Software Design Description (SDD), a Software Verification and Validation Plan (SVVP), a Software Verification and Validation Report (SVVPR), user documentation and a Software Development Plan (SDP) should be required by the SQAP. Table 2 gives a brief description of each of these plans and reports as defined in ANSI/IEEE 610.12 and ANSI/IEEE 983 [8][12].

Management planning of an effective software quality assurance effort should include a determination of objectives following performance of a criticality analysis. The criticality of an item of software determines how many of V&V tools will be needed during

each phase of software development. FIPSPUB 132 and ANSI/IEEE 983 discuss use of a criticality study. Management and reporting of the V&V effort is discussed in FIPSPUB 101, ANSI/IEEE 983 and ANS 10.4. Ongoing management reviews and audits during the development life cycle and reporting of test results are recommended by these publications. As a minimum the referenced commercial standards recommend that a Software Requirements Review (SRR), a Preliminary Design Review (PDR), a Critical Design Review (CDR), and a Software Verification and Validation Plan Review (SVVPR) be required by the overall plan. This is in addition to the normal functional and physical audits conducted as a part of the software development. Life cycle maintenance can also be extended to cover software updates. The reviews listed are briefly described in Table 3 based on definitions taken from ANSI/IEEE 610.12.

TITLE	DESCRIPTION
Software Requirements Review (SRR)	A requirements review is specified for one or more software configuration items to determine whether they form a good basis for proceeding into preliminary design after consideration of system requirements. <i>DOD name for this review is a Software Specification Review</i>
Preliminary Design Review (PDR)	"A review conducted to evaluate the progress technical adequacy, and risk resolution of the selected design approach for one or more configuration items; to determine each design's compatibility...;to determine the degree of definition and assess the technical risk associated with the selected manufacturing methods...; to establish the existence and compatibility of functional interfaces..."
Critical Design Review (CDR)	"A review conducted to verify that the detailed design of one or more configuration items satisfy specified requirements".
Software Verification and Validation Report (SVVR)	"The SVVR shall describes the results of the execution of the SVVP. This shall include the results of all reviews , audits and tests".

Table 3.

Continuing to use standardized processes when constructing software revisions is important because the programming staff used during this effort is often different from the staff which originated the project. This reduces the chance of deviating from the original concept used when the software was initially formulated.

DOD-STD-2167 and DOD-STD-2168 do not specifically require structured programming or V&V but do require a SDP. DOD instruction 5000.2 mandates the use of these standards for computer resources used in weapons systems, weapons system training, simulation testing and maintenance. The instruction requires the inclusion of structured programming, top down design and V&V techniques in any SDP generated under these standards. An independent V&V effort separate from the development team is specifically required.

Thus far we have only discussed software quality assurance publications. Software safety is mentioned only in passing references in these publications, often in the context of ensuring that critical software functions happen as required. However, as previously stated, safety is a function of determining what you don't want to happen, and ensuring that it does not. At present, only Military Standard 882B, "Military Standard System Safety Program Requirements", officially addresses this topic. A draft IEEE software safety standard is now in the review process but not yet available for use.

The 300 series tasks of MIL-STD-882B are specifically designed for software safety. These tasks are called out to be used in concert with DOD-STD-2167 Defense System Software Quality Evaluation, MIL-STD-483A, MIL-STD-1521B and DOD-HDBK-287 Defense System Software Development Handbook when complicated software packages are being constructed.

After the potential danger conditions are identified, the system design is modified to minimize the hazard. These design changes become requirements and become part of the V&V process. That the safety process is linked to the V&V process immediately becomes evident when the 300 series tasks are examined.

A top-level design hazard analysis, detailed design hazard analysis and code-level software hazard analysis are required by Tasks 302, 303 and 304. Software safety testing is required by Task 305. Testing must verify that the software functions safely both within specified environment and under abnormal conditions. Abnormal conditions testing and system integration and acceptance testing are specifically required by this standard. Task 306 requires the contractor to perform and document a Software User Interface Analysis. The development of software users procedures is also required.

If the program effort is less complicated or if for some reason the 300 Series tasks were not used, portions of the 200 Series tasks apply. Specifically, Task 202 (preliminary hazard analysis) requires a study of interfaces and the contribution of software to system mishaps. Task 203 (Subsystem Hazard Analysis) has a large proportion of the task description concerned with the potential contribution of software to hazardous subsystem occurrences. The contribution of software and its associated interfaces to system hazards are also required to be evaluated in the system and operating system analysis in Tasks 204 and 205.

To summarize, there are many commercial and government standards that relate to software quality assurance. Software safety standards are less established in the commercial

environment. All of these standards are really management guides. When to employ these standards is of critical importance. On government projects standards are sometimes mandated. A discussion of when standards are best used is the basis of the next section.

Invoking Software Standards and Acquisition Management Techniques

There are a variety of commercial and governmental software standards. Standards are of little use if they are not invoked. Commercial standards are used by the manufacturer to provide a buyer a measure of confidence that the software has been constructed properly. There is no requirement to use standard techniques except for the insistence of the buyer. DON safety regulations, as defined in OPNAV Instruction 5100.24A [22], stipulate that the system safety analysis defined in MIL-STD-882B is required to be used for all system and facility acquisitions which are Acquisition Category (ACAT) I and II Programs. An ACAT II Program has a threshold of 500 million dollars production cost while an ACAT I Program has a one billion dollar threshold. DOD safety requirements which are listed in DOD Instruction 5000.36 [23] direct establishment of system safety programs and application of MIL-STD-882B for each system acquisition. It also requires application of system safety programs for off the shelf procurements based on the mishap potential. Any of the government standards can be invoked by the Acquisition Manager if desired.

In many cases, even in government procurements, quality and safety standards are not required to be used. In the example of the computer-controlled crane which was discussed in the introduction, none of the standards mentioned in this paper are required because the total cost of the crane is below the threshold of an

ACAT II Program, and because the crane purchase was not directly connected with weapon system maintenance. This is true even though the material lifted by the crane may be hazardous or critical in nature. In fact, the crane is not required to be designed to meet any of the commercial or government software standards [24][25]. In discussions I held with the chief control engineer and the primary acquisition team that neither were aware of the existence of any of the previously highlighted safety and software quality assurance standards. It would seem reasonable that a system hazard analysis similar to what is called out in Task 204 of MIL-STD-882B probably should have been conducted. No formal documented safety analysis of this crane was carried out during its procurement. However, a safety review of crane operations was conducted by the receiving activity prior to acceptance of the crane.

The design and procurement officials were very careful to ensure that all dynamic and structural characteristics of the crane were specified and followed. It is the author's belief that often engineers and acquisition managers who are familiar with analog control systems are unaware of possible difficulties associated with digital systems. Education of acquisition managers to these differences should be considered. In addition consideration should be given to changing the DON safety instruction to prevent the cost of a procurement from being the sole determining factor for invoking quality and safety standards. Instead a matrix which considers both cost and criticality should be used for determining the need for applying a given level standardized review. Such a matrix is shown in Figure 6 where the x-axis consists of increasing cost or procurement category while the vertical axis represents increasing system criticality. The

individual cells of the matrix contain the minimum MIL-STD-882B series tasks which to be invoked.

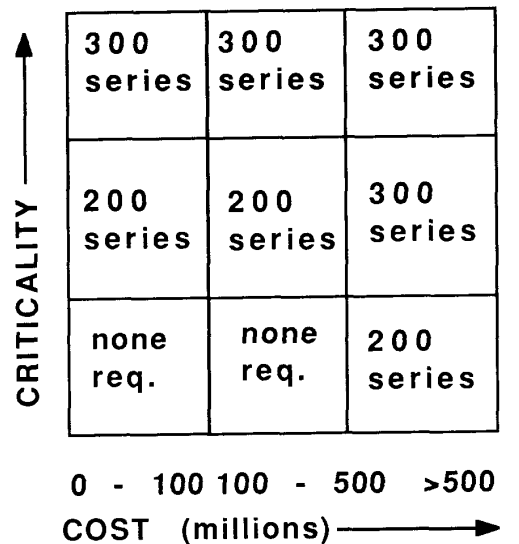


FIGURE 6

If after becoming more familiar with this subject a manager decides that an existing project needs a quality or safety review, a plan should be prepared. The V&V guide for the nuclear industry, ANS 10.4, provides guidance for software reviews for already existing commercial products. This guidance could easily be applied to any industrial application. A plan is developed under this standard results in examination of the problem statement, requirement specifications, design specifications and results of the test plan for the existing software. If this data is available, the reviewer's plan should also include a statement of why the V&V is being performed after the fact and what action is to be taken to support missing development products. The level of review depends upon the needs of the user. A similar review could be conducted to confirm system safety. For some a review of this type would be beyond the technical capability of their organization. In that

case an outside review agency should be considered. Numerous commercial software companies exist that can conduct this type of analysis. A requirement to formulate safety plans for existing critical systems which do not have them might be an appropriate change to the present DOD policy.

Conclusion

Software related systems quality and safety is best achieved by a structured management process. Commercial and government standards exist which can define how best to manage the design. In the area of safety, both cost and the criticality of the project should be considered when determining the degree of complexity required for management action. In the final analysis, the degree of quality and safety of any product especially one controlled by computer is based on the standards set by the user.

References

- [1] Leveson, N. G. "Software Safety: Why, What, and How," University of California Irvine Technical Report 86-04, February 1986.
- [2] Friend, A. W., Safety Division Naval Space and Warfare Systems Command, Discussions on Software Safety, 4 March 1991.
- [3] Military Standard 882B, "System Safety Program Requirements", 30 March 1984, US Department of Defense.
- [4] Shooman, M. L., Software Engineering Design, Reliability and Management, McGraw-Hill Book Company, 1983.
- [5] Dunn, R. H., Software Defect Removal, McGraw-Hill Book Company, 1984.
- [6] "Guideline for Lifecycle Validation, Verification and Testing of Computer Software", FIPSPUB101, National Bureau of Standards, Gaithersburg, MD 20899, 1983.
- [7] "Guideline for Software Verification and Validation Plans", FIPSPUB132, National Bureau of Standards, Gaithersburg, MD 20899, 1987.
- [8] ANSI/IEEE Std. 610.12-1990, "Glossary of Software Engineering Terminology", The Institute for Electrical and Electronics Engineers, Inc., New York, NY 10017, 1990.
- [9] ANSI/IEEE Std. 730-1989, "Software Quality Assurance Plans", The Institute for Electrical and Electronics Engineers, Inc., New York, NY 10017, 1989.
- [10] ANSI/IEEE Std. 828-1990, "Standard for Software Configuration Management", The Institute for Electrical and Electronics Engineers, Inc., New York, NY 10017, 1990.
- [11] ANSI/IEEE Std. 829-1983, "Standard for Software Test Documentation", The Institute for Electrical and Electronics Engineers, Inc., New York, NY 10017, 1983.
- [12] ANSI/IEEE Std. 983-1986, "Guide for Software Quality Assurance Planning", The Institute for Electrical and Electronics Engineers, Inc., New York, NY 10017, 1983.
- [13] ANSI/IEEE Std. 1012-1986, "Standard for Software Verification and Validation Plans", The Institute for

Electrical and Electronics Engineers, Inc., New York, NY 10017, 1986.

[14] ANSI/IEEE Std. 1016-1987, "Recommended Practice for Software Design Descriptions", The Institute for Electrical and Electronics Engineers, Inc., New York, NY 10017, 1987.

[15] ANSI/IEEE Std. 1028-1988, "Standard for Reviews and Audits", The Institute for Electrical and Electronics Engineers, Inc., New York, NY 10017, 1988.

[16] ANSI/IEEE Std. 1042-1987, "Guide to Software Configuration Management", The Institute for Electrical and Electronics Engineers, Inc., New York, NY 10017, 1987.

[17] AFSC/AFLCP 800-5 Air Force Systems Command and Air Force Logistics Command Software Independent Verification and Validation, Washington, DC, 1988.

[18] ANSI/ANS 10.4-1987, "Guidelines for the Verification and Validation of Scientific and Engineering Computer Programs for the Nuclear Industry", American Nuclear Society, La Grange Park, IL 60525, 1987

[19] DOD-STD-2167A Military Standard Defense System Software Development, Department of Defense, Washington, DC, 1988.

[20] DOD-STD-2168 Military Standard Defense System Software Quality Program, Department of Defense, Washington, DC, 1988.

[21] Wallace, D. R. and Fujii, R. U., "Software Verification and Validation: Its Role in Computer Assurance and Its relationship with Software Project Management Standards", NIST Special Publication 500-165, September 1989.

[22] OPNAV Instruction 5100.24A, "Navy System Safety Program", US Department of the Navy.

[23] DOD Instruction 5000.36, "System Safety Engineering and Management", US Department of Defense.

[24] Gedgard W, Northern Division Naval Facilities Engineering Command, Telephone conversation of 15 April 1991.

[25] Department of the Navy, Northern Division Naval Facilities Engineering Command, Request for Technical Proposal No. N632472-82-12-1455 of 4 October 1984.