Software Quality Management II Vol. 1:

Setting Up a Pre-production Quality Management Process in the Medical Device Industry

S. B. Leif,[a], S. H. L. Aha[b] & Robert C. Leif [a]

Ada_Med 5648 Toyon Rd, San Diego, CA 92115.

[a]Tel.(619)582-0437, Email rleif@rleif.com

[b]S. H. Leif, Email steph@newportinstruments.com

ABSTRACT

Medical device development and manufacture is a large, regulated industry. Medical devices now include a substantial software component which must be developed according to "good manufacturing practices." Good manufacturing practices require the definition and enforcement of software processes which ensure quality products by defining a detailed method of software construction. Although the technical problems of creating a software process are significant, the political and human problems associated with its implementation are of greater difficulty. Our experience with organizing the software process including the transfer software technology from the defense industry to medical electronics will be discussed.

INTRODUCTION

This paper is based on our experience with instrument development and manufacturing for the medical device industry. These observations should be applicable to other large, regulated industries and any industry which requires the minimization of software defects. Medical devices including the associated software are regulated by the Food and Drug Administration (FDA) in the United States and many nations require that device manufactures adhere to ISO (International Standards Organization) 9000.

The FDA is responsible for assuring the safety and effectiveness of medical devices under the 1976 Medical Device Amendments to the Federal Food, Drug, and Cosmetic Act. In November of 1990, the US Congress passed the Safe Medical Device Act, which extended their authority to the pre-production area. The FDA expects to see a defined process,

proof that the defined process was followed, as well as an established quality system. Pre-production is all activities that take place before manufacturing. For software, production is disk duplication or other forms of generating distributable electronic media. Thus, pre-production activity encompasses everything in the software life-style except for maintenance.

The FDA requires pre-market notification before a medical device can be marketed and sold. For new technology, the process is called Pre-Market Approval; for enhancements and new instruments which are substantially equivalent to ones already being marketed, the submittal is in the form of a 510(k). In 1987, the FDA released in draft form a "Reviewer Guidance for Computer-Controlled Medical Devices". This document [1] defines the kind of information on software that FDA reviewers expect to find in a 510(k) or Pre-Market Notification. Included in this document is the requirement that the submission must contain a certification which is a:

> "Written affirmation stating that software development was followed, that good quality assurance procedures were adhered to, and that test results demonstrate that the system specifications and functional requirements were met [1]."

Many medical devices include a substantial software component, which according to regulatory agencies must be developed according to "good manufacturing processes".

The FDA started to monitor device recalls during the mid 1980's. Their studies showed that 45% of all recalls were due to preproduction related problems. These problems were directly traceable to one or more of the following: poor design, a lack of or inadequate requirements, or failure to follow a documented life cycle.

ISO has promulgated Quality Management and Quality Assurance Standards (REFS). The European Community and many other countries will employ the ISO standards. Some countries are already starting to require ISO certification for all medical devices. Companies interested in world wide sales must satisfy both ISO and FDA regulations.

There are minor differences between the requirements which are specified by the FDA and ISO. The FDA is more interested in hazard analysis and does not require any economics of conformance. Economics of conformance is the total cost of making, finding, repairing or avoiding defects. It is made up of prevention cost, appraisal cost, internal failure cost, and external failure cost. In assuring compliance, the ISO certifica-

tion is done by a third party, while the FDA relies upon the word of the company. If the FDA finds that the company falsified statements those responsible can be imprisoned on felony charges and the company fined.

GOOD MANUFACTURING PRACTICE FOR SOFTWARE

The key elements that the FDA and ISO look for in the software area are:

1) A documented and adhered to software development process.

2) Software requirements, hazard and safety concerns in the requirements.

3) Hazard and safety traceability through design, code and testing.

4) Formalized testing with evidence through traceability that the hazards and safety issues were addressed.

Good manufacturing processes [2] start with the design input. Both agencies expect to see a documented software life-style with emphasis on specifying requirements. The FDA expects companies producing software for medical devices to have in-house standards for programming. These standards should specify the requirements and restrictions which are to be followed when writing computer programs. The standards should cover documentation procedures, coding standards, and software testing standards. The software should be written in modules which can be tested. The testing process must be comprehensive and well documented. The hazard analysis must be thorough and should be reiterated for each development stage. Wherever possible, the system should include fail safes. The code should be inspected and reviewed to ensure that it reflects the design and follows the company policies.

Once requirements for the software are determined, the software according to the US Food and Drug Administration and ISO 9000 must be produced following good manufacturing practices. Unfortunately these organizations have not provided a simple description of software good manufacturing practice.

A quality system must be compliant with the regulations, yet in order to be effective, be easy to follow and documented. The means for monitoring processes must be unobtrusive yet accurate. The essential elements of the quality system are: configuration management, verification and validation, quality assurance, and project management.

The first step in developing a documented software process is to specify the overall process. This can be done with a Software Project Development Plan.

Software Project Development Plan

This plan specifies the software development process for all medical device software projects. It states how software projects are to be managed and controlled by a manufacturer and the software development life cycle. The key elements to a plan are:

1) a list of the required documents,

2) the order in which they will be developed,

3) which group or type of individual is responsible for creating each of the required documents,

4) the position of the individuals who review and approve each of the documents.

A good technique is to write a plan to permit Boehm's spiral development model [3]. Boehm's model is much closer to the reality of software development for instruments, as has been previously described [4] than the standard waterfall model.

The Software Project Development Plan should cover controlling and describing a project. The key controlling documents are:

(1) the Software Project Management Plan [5],

(2) the Software Quality Assurance Plan [6],

(3) the Software Configuration Management Plan [7], and

(4) the Software Verification and Validation Plan [8].

These plans are generic and are tailored to each project. They do not need to be large and overwhelming. Depending upon the project, they can be quite concise.

Key Project Plans

Software Project Management Plan This document specifies the plan for control and management of the elements of a software project. It defines the technical and managerial processes necessary to satisfy the project requirements. This plan needs to be written at the inception of each software project; however, the basic outline should apply to all projects. When

the plan is completed, it must be given a management review to insure compliance with the organization's established software development process.

As the project proceeds, this plan must be reviewed and updated at the conclusion of each phase of the software development life cycle. The Software Project Management Plan details the organizational structure, organizational boundaries and interfaces as well as project responsibilities. This document provides the forum to present management objectives and priorities, as well as stating what risks have been identified and what assumptions have been made. It is also the place where the work packages, schedule and budget are presented.

Software Quality Assurance Plan This document specifies how the project shall be monitored, and reviewed. In addition, this plan will ensure that the software development process is in compliance with the Software Project Development Plan. This plan establishes a mutual understanding between software development and software quality assurance personnel, with respect to how the software development process will be monitored.

Software Configuration Management Plan This document provides the basis for keeping a project under control. It delineates the authority and responsibility for entering new elements under control, updating and editing existing elements and insuring that unauthorized changes do not occur. This plan applies to the development and maintenance of all the specifications, designs, test plans, source code, object code, test data, and documentation. This plan also applies to the maintenance of supporting software products and tools used in the development of software.

Key elements that the Software Configuration Management Plan should include are: a methodology for identifying the configuration elements, configuration control, interface control, as well as a way to do configuration status accounting. This plan should also identify the management, authority and responsibilities of the software configuration manager. Software tools are available to assist in this process. At least, one language, Ada, will not permit compilation of components in an incorrect order. The capacity to reliably control multiple versions of the software is required. Detection of obsolete components is available in Ada.

Software Verification and Validation Plan This document must be started early in the project. It evolves as the project matures. The first elements which are covered in the early plan are the acceptance testing and system testing. As the project develops, the integration test plans and module test

plans are added. The purpose of a Software Verification and Validation Plan is to describe the process of determining whether the software requirements are complete and correct, the products of each software development phase fulfill the requirements or conditions imposed by the previous phase, and that the final system complies with specified requirements.

The Software Verification and Validation Plan will help ensure that errors are detected and corrected as early as possible in the software life cycle and that project risk, cost, and schedule effects are lessened. The plan can enhance software quality and reliability as well as system safety. Management visibility into the software process is improved. Proposed changes and their consequences can be assessed quickly. The plan also provides checks that the software Verification and Validation process is in compliance with the software development process.

The product items covered by a software verification and validation plan should include all software that is considered part of the instrument and all hardware related to fail-safe requirements. Each system hazard identified in the specifications should have compensating fail-safe requirements associated with it. The plan should cover the verification and validation activities that will be performed.

The goals of a verification and validation effort are to identify problems, as early as possible, in the product development life cycle, report them to the project team for correction, and to perform verification and validation tasks in parallel with the project development effort. This will enable the project to ensure compliance with regulatory requirements. The early development of a verification and validation plan will facilitate the performance of software testing at the component, integration, system, acceptance test level, and hazards testing. Documentation of the hazards testing activities, including traceability between the applicable documents, will be easier, as will the performance of regression testing, as required to verify corrective action to code. The ultimate goal is to release project software which is safe and meets the specified requirements.

Specific Project Documents

Describing a specific project requires the following specific documents:

(1) Concept or Product Specification,

(2) System Specification,

(3) Software Requirements Specification,

(4) Software Design,

(5) Hazard Analysis,

(6) the code itself.

Concept Specification or Product Specification This document is the launch of a new product. In many organizations, it is written by the marketing department and refined with the consensus of the engineering staff. This document describes what is wanted in terms of marketplace, capabilities, cost, and performance. Approval needs to be given by: the corporate financial body for permission to develop and build; the engineering department in terms of feasibility, cost and delivery; manufacturing, service, and sales also need to have input.

System Specification The System Specification is generated from the Concept Specification. This documents details in rough strokes the major components of the system in terms of electrical, mechanical and software. It adds detail to further describe the "what" of the system.

Software Requirement Specification This document describes the detailed requirements for the software. The Software Requirement Specification is best done as a joint effort between marketing, engineering, service, and manufacturing. It contains the user interface for the instrument. Communication between the groups can be a major problem. Most non-software personnel do not have an understanding of software engineering or the capability to write software specifications. Most CASE tools are incomprehensible to marketing professionals. One simple solution that has been successfully employed is to simulate the system by story boarding the screens [9] [10]. This is facilitated by employing commercially available screen generators [11] [12] [13]. Often, the code produced by the generators can be used in the final product. Unfortunately, none of the major CASE tools has a direct link to screen a generator. A data dictionary of types and their ranges must be agreed upon.

The Software Requirement Specification must specify program behavior in three bounded intervals: prior to program start-up, normal program execution, after shutdown.

Before program start-up, if the hardware cannot retain or indicate receipt of an input prior to start-up, specify that inputs will be ignored. If the hardware can observe existence of an input, the specifications must specify what will be done about it. The requirements also need to specify

what will happen to inputs that arrive after shutdown.

Another area where the requirements must be specific is in the Input Capacity vs. Output Capacity. The input environment may be able to generate more inputs than output environment can absorb. The software needs to handle 3 cases:

1. Input and output rates within limits, the “normal response”.

2. Input rate within limits but output rate would be exceeded if normally timed output produced, then delayed response required.

3. Input rate excessive, then abnormal response required.

Where input and output capacities differ, there must be multiple periods for which discrete capacity assumptions are specified. For these overload conditions the requirements need to specify how they should be handled. For example, by generating warning messages, generating output to reduce load (i.e. messages to external systems to “slow down”), locking out interrupts for overloaded channels, produce outputs that have reduced accuracy and/or response time requirements. (Graceful degradation), or by reducing the functionality provided by system.

Hazard Analysis Hazardous conditions can be the results of hardware failure, software failure, the operator, or system design. In doing a hazard analysis, all types of failures should be analyzed. Where possible, the hardware and software should be designed in a fail-safe fashion. A hazard analysis should include the probability of the condition occurring and the severity of the occurrence. This analysis should be started with the inception of the project. If potential hazardous conditions are recognized early many times they can be designed ’out’ of the system. The hazard analysis needs to be continuously updated through out the process. The final testing of the product should include all fail-safes. Sometimes, the only means of prevention is through labeling.

Software Design The software design needs to show traceability to the Software Requirement Specification. Having complete requirements up front facilitates the design of the code. The software design must respond to all of the hazardous conditions which have been identified in the hazard analysis and have been determined to be software preventable.

Both the Software Requirement Specification and Hazard Analysis are key deliverables, which must be included for the FDA and ISO. These bodies are looking for these documents to be created at the beginning of a project and updated throughout the process. Retrofitting at the end of the project is not acceptable. The FDA specifically wants to see traceability

relating requirements, hazards, design, and testing.

This requirement for traceability provides a major problem with today's CASE tools. CASE tools are best suited for water-fall based development. Real-world designs [4] tend to evolve according to Boehm's spiral model [3]. Although some CASE tools have limited capabilities to reverse engineer the drawings from code, none can after correction of the new drawings regenerate either the complete original code or that part which is still consistent with the CASE drawings. The utility of CASE tools will be greatly increased when they and the source code are really two coupled views of the same object and are directly linked. It should also be noted that individuals educated in languages based on alphabets can have difficulty with representations based on pictographs or hieroglyphics.

Code The code produced should be written to a standard. The code must be inspected and have been reviewed to ensure that it reflects the design and has followed the company policies.

EXPERIANCES

The introduction of a quality plan which requires the creation and maintenance of the documents described above is a very serious undertaking. We will describe efforts to develop and implement such a plan. Initially, the development process which is followed is often established by tradition and undocumented. An evaluation using the Carnegie Mellon Maturity Model for Software [ref] often results in a score of level one. Approach

What the engineers were actually doing is often fairly close to what is required. The problem is to document their present procedures and enforce consistency between projects. Each element of the process must be written as a task with a thumbnail description of the requisite inputs and outputs and the responsible parties designated. All of these tasks are then linked graphically to show parallel activities, precursors, and successors and documented in a Process Guideline.

The major problems in implementing the process are often political rather than technical. Upper management is often afraid to commit the organization in writing to explicit procedures, because that would mean they would be required to follow them. Engineers see documentation as requiring more paperwork. Their usual refrain will be "what do you want,

a product, or paperwork?”

A Standard Operating Procedure often states that development will sequentially follow a process of Feasibility, Design, Prototype, Pilot, and Manufacturing (an explicitly waterfall process). A key requirement is that each project must write a project plan which documents the applicable tasks from the Process Guideline. Projects are required to write a project plan which would cover all of the elements of a quality system such as project management, verification and validation, quality assurance, and configuration management.

The problem is to provide consistency with some degree of flexibility. Project managers must not be given freedom to pick and choose what tasks that they feel like doing. A good check and balance of the system is that the plans must be approved by the Director of Product Development.

Our solution is to follow the IEEE guidelines [14] in developing the documentation described above and to employ the Ada programing language.

Training and Implementation Although it is important to make the engineers cognizant of the formal process, one should avoid painful processes that instill hostility. In one case, management required the staff to fill out time-cards using tasks from the Process Guideline. This forced the engineers to read the Process Guidelines and relate their activities to the formal process. A more unobtrusive way to accomplish the same end is to encourage the engineering staff to take software engineering courses. The more interested individuals will become actively involved in developing the process and serve as evangelists for the rest of the organization.

Software engineering courses including the study of the Ada programming language often are quite upsetting to some members of the staff. Sometimes, the exposure to this material induces the most vociferous opponents of a codified process to seek employment elsewhere.

The process must become an accepted part of corporate culture. It is important to have frequent feedback to measure compliance to the process. If the process is not being followed then management must determine if it is due to a lack of understanding/education, the process is too complex, or the engineers lack of understanding that this is what upper management really wants. It is very easy to write procedures which are overly complex and impossible to follow. The feedback should act as a

monitor against this. Do not be upset if procedures need to be refined often in the beginning. In order to obtain the full benefits of process in a corporation the process must be embraced enthusiastically by both management and the users.

Change will always receive a mixed reception. Some will denounce it as a strait jacket that hinders their creativity. Others will praised it. Most people want to know what is expected of them. A process guideline can be of great help in ensuring that the proper prerequisites are completed. Many times in the past, we have observed that tasks such as prototype manufacturing were started prematurely. The Process Guideline works as a checklist to ensure that long lead tasks are started in a timely fashion.

Upper Management

The technical work of developing the process can be overwhelmed by the social consequences of trying to implement the process without the complete support of management. A critical element towards guarantying success is the buy-in of upper management. In several cases, when upper management was not totally committed, attempts towards codifying the process, were undermined by the nay-sayers. What we did not do is bill the new process like we would a new project and have the engineers sign on. However, this would have been impossible until engineering management signed on to the new process

TRANSFER OF DEPARTMENT OF DEFENSE TECHNOLGY TO MEDICAL DEVICES

Ada Programing Language

Advantages of Ada Ada is a language which has been designed based on software engineering concepts and has been successfully employed in systems of even greater criticality than medical devices [15]. Ada is a portable, general purpose language. The United States Defense Department requires that any Ada compiler used internally or on a government contract be validated. For this purpose, it has supported and maintained a validation suite which assures with some degree of confidence that compilers operate and that the Ada standard will hold across different hardware configurations. Of course, code that interacts directly with hardware will have to be changed to reflect the new hardware. Ada code is constructed out of packages, which only interact through their specifications. A special "with" statement must be included in a package to make specifications of other packages visible. The package bodies, which are the parts that actually perform the operations, are invisible to the other packages. Hardware

dependent changes are made whenever possible in the bodies. This deliberate limitation on visibility controls the interactions between packages. These interactions are the side effects that plague software development and testing. The package structure provides the modularity required for a group to develop code.

Ada is readable and self-documenting. Safety features include: strongly typing, range checking, and exceptions. Real-time constructs based on tasking are included in the language. Ada compilers are like most equipment employed for the manufacture of medical devices come with a significant warranty.

Established Methodology and Software Engineering Culture The requirements of the Defense Department have resulted in the establishment of independent verification and validation organizations. The military, NASA and FAA have developed the technology to maximize both safety and productivity. Well engineered code is often available from software repositories. Since the Department of Defense has mandated the use of Ada for all new software including management information services, a large selection of software tools is available to assist in producing software documentation. Since Ada is a product of the software engineering establishment, its practitioners often see the attempt to achieve zero defects as a moral booster and a professional obligation.

Project Management The Ada package structure permits partitioning of the software into manageable components. These packages are easily traceable to the software requirements and are the basic structuring mechanism for the software system. The software project schedule throughout the development life-cycle can be monitored in terms of the Ada packages. The separate definition of interfaces (specifications) and their implementations (bodies) in Ada allows an orderly integration of the software components and reduces the integration time.

Hazard Analysis Ada includes an exception construct, which is employed to handle abnormal situations and permit recovery of the system. Many of the hazards can have a one to one correlation with an exception.

Testing The white box testing that occurs during verification and validation includes checking the behavior of the system by entering a range of values for each of the variables entered by the user or that could be corrupted by the system. Since Ada compilers permit data types to have specific ranges and the initialization of variables when they are declared, coverage for a variable can be achieved with fewer cases than for a weakly typed language, such as C. Ada compilers produce error messages when a potential over-range condition occurs for a data type.

The presently published data indicates that Ada is still safer [16] and more cost effective than any third generation language now known [17]. The use of Ada for a medical device has been demonstrated [18]. For a medical device, the benefits of Ada 83 outweigh the disadvantages. The syntax and capabilities of Ada 83 are well suited for the efficient development of the quality expected for a medical device.

Ada 1994 Object oriented and necessary low level features have been added. Ada 94 permits object oriented programming with effiency and safety[21]. Because of its many built in software engineering features described above, Ada is a language of choice for a medical device [19] [18].

CONCLUSION

The software process is an important way to ensure software quality. But the quality of the software processes themselves will determine their effectiveness. For a company which is regulated, one of the worse things to do is to write elaborate procedures and then not follow them. In developing processes, it is best to make the first iterations very general, gather experience in following the general procedures, and then gradually refine them. The development of a software process is similar to the development of any large software project [22]. A model, such as Boehm's spiral model [3], which supports successive refinements is preferred to the inflexible waterfall model.

Paradoxically, the requirements of the regulatory authorities, besides improving the product, can decrease the overall cost. A complete specification and well documented design will speed product introduction by permitting the use of concurrent engineering techniques and minimize the time for review by the regulatory agencies. A well defined software process will result in fewer errors and facilitate maintenance including enhancements. The construction of a library of tested, reusable software components (objects) will reduce both costs and time to market. Modern modular languages, such as Ada, are designed to achieve this goal.

The choice of programming languages is important from a quality point of view because programming languages are like any other manufacturing tool or material. The use of tools of known quality also fosters pride in a developers' workmanship. The presently published data indicates that Ada is still safer [16] and more cost effective than any third generation language now known [17] and Ada'94 offers an excellent solution

to the problems associated with object oriented programming. We do not understand why medical device manufacturers continue to develop software using old technology such as C and FORTRAN.

Standardized processes which everyone follows benefits management because they have a consistent measuring stick on which to gage progress. This standard process does not imply that individual groups can no longer innovate. Instead it should foster the transfer of innovations from one group to corporate wide. While the processes discussed in this paper have primarily been document driven, we see a movement towards object driven processes in the future. We hope to transfer some of the good ideas behind the object-oriented bandwagon to process.

REFERENCES

1. "Reviewer Guidance For Computer Controlled Medical Devices Undergoing 510(k) Review," Office of Device Evaluation Center for Devices and Radiological Health Food and Drug Administration, Department of Health and Human Services, Public Health Service, Stamped Aug 29, 1991.

2. Leif, S. B., and Leif, R. C. "Producing Quality Software According to Medical Regulations for Devices", Computer Based Medical Systems, Proceedings of the Fifth Annual IEEE Symposium 265-272, 1992

3. Boehm, B. W. "A Spiral Model of Software Development and Enhancement," *Computer,* May, p.61, 1988.

4. Leif, R. C., Leif, S. B., and Leif, S. H. "The Real World of Instrument Development". *IEEE Symposium Record, Policy Issues In Information and Communication Technologies in Medical Applications,* IEEE Catalog No. UH0181-8 (1988).

5 ANSI/IEEE Std. 1058.1-1987 IEEE Standard for Software Project Management Plans.". Institute of Electrical and Electronics Engineers, Inc., New York, New York.

6. ANSI/IEEE Std 730-1989 IEEE Standard for Software Quality Assurance Plans. Institute of Electrical and Electronics Engineers, Inc., New York, New York.

7. ANSI/IEEE Std 828-1990 IEEE Standard for Software Configuration Management Plans. Institute of Electrical and Electronics Engineers, Inc., New York, New York.

8. ANSI/IEEE Std 1012-1986 IEEE Standard for Software Verification and Validation Plans. Institute of Electrical and Electronics Engineers, Inc., New York, New York.

9. Leif, S. B., Leif, R. C., and Auer, R. “The EPICS® C, an ergometerically Designed Flow Cytometer Computer System.” *Analytical and Quantitative Cytology and Histology,* vol. 7 p. 187, 1985.

10. Andriole, S. J. “Rapid Application Protyping, The Storyboard Approach to User Requirements Analysis, Second Edition”, QED Technical Publishing Group, Wellesley, MA, 1992

11. AdaSAGE

12. Downs, M., Duff, J., Mackey, K., Teyssier, L., Tonas, C. “Using X with the Ada Mind-set”, Conference Proceedings Tri-Ada ’93 20-28 1993.

13. TeleUse

14e. “IEEE Software Engineering Standards Collection, Spring 1991 Edition,” Institute of Electrical and Electronics Engineers, Inc., New York, New York.

15. Pyle, I. C. *Developing Safety Systems, A Guide Using Ada*, Prentice Hall ISBN 0-13-204298-3.

16. Tang, L. S. “A Comparison of Ada and C++”, Conference Proceedings TRI-Ada ‘92, 338-349, 1992.

17. Mosemann, L. K. “New is Good--Is Ada Too Old”, CrossTalk, The Journal of Defense Software Engineering, 40 8, 1993.

18. Leif, R. C., Sara, J., Burgess, I., Kelly, M., Leif, S. B., and Daly, T. “The Development of Software in the Ada Language for a Mid-Range Hematology Analyzer”. Tri-Ada ‘93 340-346 (1993).

19. Leif, R. C., Rosello, I., Simler, D., Garcia, G. P., and Leif, S. B. “Ada Software for Cytometry,” *Analytical and Quantitative Cytology and Histology,* vol. 13 p. 440, 1991.

20. Taft et al.T. “Ada 9X Mapping Specification 4.6 & Rationale 4.1”, Intermetrics Inc. Cambridge MA, 1992.

21. Banner, B., and Schonberg, E., “Assessing Ada 9X OOP: Building a Reusable Components Library”, Conference Proceedings TRI-Ada ‘92, 79-90, 1992.

22[Osterweil] Leon Osterweil, L., “Software processes are software too” 9th International Conference on Software Engineering, 1987.

23. Office of Device Evaluation Center for Devices and Radiological Health Food and Drug Administration, Department of Health and Human Services, Public Health Service. “Reviewers Guidance for Computer Controlled Medical Devices Undergoing 510(k) Review”, Stamped Aug 29, 1991.

24. [2] “ANSI/ASQC Q90-1987 American National Standard, Quality Management and Quality Assurance Standards--*Guidelines for Selection and Use”, American Society for Quality Control,* Milwaukee, Wisconsin, 1987.

25[3] DOD-STD-2167A, DEPARTMENT OF DEFENSE, WASHINGTON, D.C. 20301, Defense System Software Development

26. “Reference Manual for the Ada Programming Language,” Ada Joint Program Office, The Pentagon, Washington, D.C. 20301