Guidance for FDA Reviewers and Industry

Guidance for the Content of Premarket Submissions for Software Contained in Medical Devices

Document issued on: May 29, 1998

U.S. Department Of Health And Human Services
Food and Drug Administration
Center for Devices and Radiological Health
Office of Device Evaluation

Preface

Public Comment

Comments and suggestions may be submitted at any time for Agency consideration to, Joanna H. Weitershausen, HFZ-450, 9200 Corporate Boulevard, Rockville, Maryland 20850. Comments may not be acted upon by the Agency until the document is next revised or updated. For questions regarding the use or interpretation of this guidance contact Joanna H. Weitershausen at 301-443-8609 or by electronic mail at JXH@CDRH.FDA.GOV.

Additional Copies

World Wide Web/CDRH home page: http://www.fda.gov/cdrh/ or CDRH Facts on Demand at 1-800-899-0381 or 301-827-0111, specify number 337 when prompted for the document shelf number.

Table of Contents

List of Figures and Table

SECTION 1. Introduction

1.1 Purpose

This document provides guidance on the regulatory review of premarket submissions for software contained in medical devices. It replaces the "Reviewer Guidance for Computer Controlled Medical Devices Undergoing 510(k) Review" issued in 1991. This guidance provides a discussion of the key elements of a premarket medical device software submission; thereby providing a common baseline from which both manufacturers and scientific reviewers can operate. This guidance document represents the Agency's current thinking on premarket submissions for medical devices containing software. It does not create or confer any rights for or on any person and does not operate to bind FDA or the public. An alternative approach may be used if such approach satisfies the requirements of the applicable statute, regulations or both.

1.2 Background

This document has been developed in response to requests to refine and clarify the 1991 software guidance, based on feedback from both manufacturers and scientific reviewers. By clarifying the 1991 software guidance, the Agency hopes to receive a larger percentage of complete premarket submissions without the need for additional information requests which are time and resource consuming for both FDA and manufacturers. In addition, this document has been updated to be consistent with emerging international consensus standards such as International Electrotechnical Commission (IEC) 60601-1-4[1], International Organization for Standardization (ISO) 9001 and ISO 9000-3, and the Quality System Regulations, 21 CFR Part 820.

1.3 Scope

This document applies to all types of premarket submissions for medical devices containing software and for software products considered by themselves to be medical devices. This includes premarket notifications (510(k)s), premarket applications (PMAs), investigational device exemptions (IDEs), and Humanitarian Device Exemptions (HDEs).

1.4 Intended Audience

This document is intended for use by scientific reviewers within the FDA CDRH Office of Device Evaluation (ODE), the medical device industry, and other interested parties.

[1] IEC 60601-1-4 is a safety standard for programmable electrical medical systems which includes requirements for the process by which the medical electrical equipment is developed. It is not a software standard. Portions of this standard related to risk estimation do not address failures of a systematic nature (hardware design failures and all software failures). Therefore, hazards associated with risk-related functions subject to systematic failure should be managed based on the severity of the hazard resulting from failure and the assumption that the failure will occur.

1.5 Document Organization

This document contains four sections and three appendices.

- Section 1 (Introduction) describes the purpose, background, scope, intended audience, document organization, relationship to other documents, and terminology.

- Section 2 (Level of Concern) explains the level of concern determination and how it relates to the documentation in and the review of a premarket submission.

- Section 3 (Documentation in a Premarket Submission) identifies what informational elements should be included in a premarket submission.

- Section 4 (Risk Management Activities during the Software Life Cycle) discusses how the life cycle relates to risk management activities.

- Appendix A (Technological Issues and Special Topics) discusses special topics and unique technological issues currently facing the FDA and industry.

- Appendix B (Relevant National and International Consensus Standards) and C (Bibliography) cite current relevant national and international consensus standards and texts related to software engineering, quality assurance, risk management and life cycle methodologies. Reviewers and manufacturers should refer to these sources for more in-depth information. This information is not duplicated in this document.

1.6 Relationship to Other Documents

This publication is a guidance document. FDA and device manufacturers use guidance documents to provide a means, among possible others, to meet regulations and policy. National and international consensus standards, such as those cited in Appendix B, are viewed as tools for achieving and demonstrating compliance with regulations.

Since this document addresses a cross-cutting issue, it is intended to complement device specific guidance(s) by providing additional detailed information specific to software products.

Figure 1 is provided to illustrate the relationship of this document to “General Principles of Software Validation” and “Guidance for Off-the-Shelf Software Use in Medical Devices”. By following the flow diagram, it can be determined which guidance is applicable.

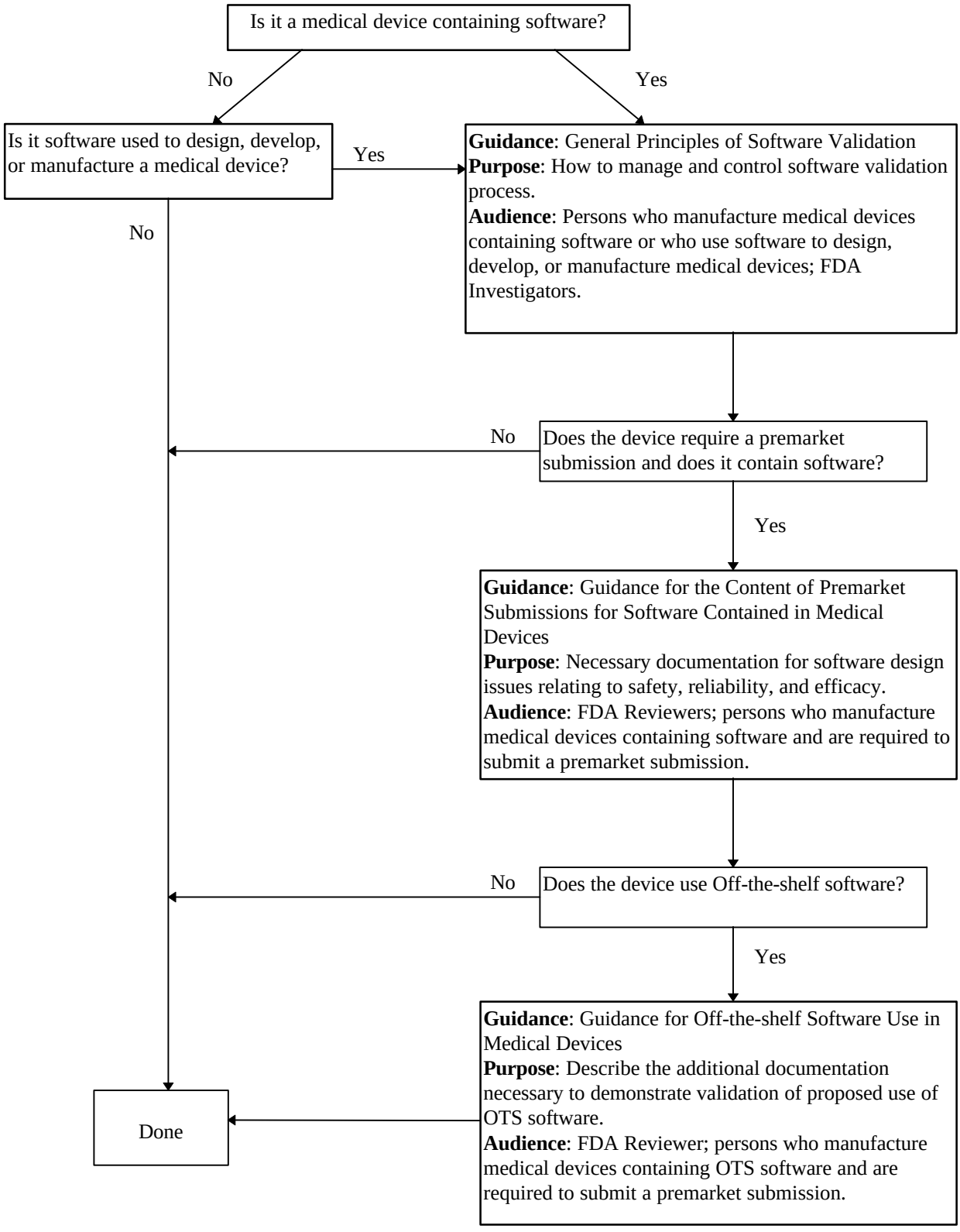

Figure 1: Relationship of Software Related Documents

1.7 Terminology

FDA/CDRH guidance documents by definition do not establish legally binding requirements. A guidance document is published to recommend a means, but not the only means, of demonstrating compliance to relevant medical device regulations.

For regulatory consistency, this document uses the terms "verification" and "validation" as they are defined in FDA's medical device Quality System regulation. However, the use of “validation” in this document is limited to “design validation” and does not include “process validation” as defined in the Quality System regulation. Unless otherwise specified herein, all other terms are as defined in the current edition of the FDA "Glossary of Computerized System and Software Development Terminology".

Verification is defined in the Quality System regulation as "confirmation by examination and provision of objective evidence that specified requirements have been fulfilled." In a software development environment, software verification is confirmation that the output of a particular phase of development meets all of the input requirements for that phase. Software testing is one of several verification activities, intended to confirm that the software development output meets its input requirements. Other verification activities include walkthroughs, various static and dynamic analyses, code and document inspections, etc.

Design validation, is defined in the Quality System regulation as “establishing by objective evidence that device specifications conform with user needs and intended use(s).” One component of design validation is software validation. Software validation is establishing, by objective evidence, that the software conforms with the user needs and intended uses of those functions that have been allocated to the software. Software validation is a part of design validation of the finished device. It involves checking for proper operation of the software, after integration into the device, in its actual or simulated use environment. Software validation is highly dependent upon comprehensive software testing and other verification tasks previously completed at each stage of the software development life cycle. For example, both alpha and beta testing of device software in a simulated or real use environment may be included as components of an overall design validation program for a software automated device.

SECTION 2. Level of Concern

FDA/CDRH uses the term "level of concern" as an estimate of the severity of injury that a device could permit or inflict (directly or indirectly) on a patient or operator as a result of latent failures, design flaws, or using the medical device software. The extent of the premarket review process pertaining to software products is proportional to the level of concern. Therefore, it is important to clarify the role of the software in causing, controlling, and/or mitigating hazards which could result in injury to the patient or the operator. Manufacturers are asked to specify in premarket submissions the level of concern for the software product and describe how the level of concern was determined. This section provides suggested criteria that could be used to establish the level of concern for medical devices containing software. This approach could also be used in determining the severity of injury that could result from each hazard identified in the hazard analysis.

2.1 Background

Inadequate or inappropriate software development life cycle and risk management activities, inappropriate use of a medical device, and/or operational errors could lead to unsafe or ineffective delivery of energy, drugs, life-supporting or life-sustaining functions, or to incorrect or incomplete information causing a misdiagnosis or selection of the wrong treatment or therapy. Therefore, risk(s) associated with potential failures or design flaws is a concern during the review of software contained in medical devices.

The level of concern for medical device software varies over a continuum. Accordingly, it is essential that the criteria used to determine the level of concern be straightforward. If level of concern determinations have already been made for specific devices and review criteria are already established by division management, division review procedures, and/or by the Office of Device Evaluation, they take precedence over manufacturers' statements of level of concern.

Various national and international consensus standards and references (Appendices B and C) classify the severity of risk on an incremental scale. This is helpful for determining the appropriate degree of rigor and the kinds of risk management activities that should be performed.

2.2 Approach Recommended by FDA for Risk Estimation and Control

In general, risk is considered the product of the severity of injury and the probability of its occurrence. However, software failures are systematic in nature and therefore their probability of occurrence can not be determined using traditional statistical methods. As such, risk estimation for software products should be based on the severity of the hazard resulting from failure, assuming that the failure will occur. Manufacturers are encouraged to use the risk identification and control techniques promoted in standards and references.

2.2.1 Definitions

The suggested approach to determining level of concern yields a major, moderate, or minor level of concern. Definitions associated with major, moderate, and minor level of concern relate to

outcomes of software failure:

Major - The level of concern is major if operation of the software associated with device function directly affects the patient and/or operator so that failures or latent flaws could result in death or serious injury to the patient and/or operator, or if it indirectly affects the patient and/or operator (e.g., through the action of care provider) such that incorrect or delayed information could result in death or serious injury of the patient and/or operator.

Moderate - The level of concern is moderate if the operation of the software associated with device function directly affects the patient and/or operator so that failures or latent design flaws could result in non-serious injury to the patient and/or operator, or if it indirectly affects the patient and/or operator (e.g., through the action of a care provider) where incorrect or delayed information could result in non-serious injury of the patient and/or operator.

Minor - The level of concern is minor if failures or latent design flaws would not be expected to result in any injury to the patient and/or operator.

Serious injury - as adopted from the Medical Device Reporting (MDR) regulation in the Code of Federal Regulations 21 CFR 803.3 (aa), means an injury or illness that:

i. is life threatening,

ii. results in permanent impairment of a body function or permanent damage to a body structure, or

iii. necessitates medical or surgical intervention to preclude permanent impairment of a body function or permanent damage to a body structure.

Permanent - for the purposes of this subpart, permanent means irreversible impairment or damage to a body structure or function excluding trivial impairment or damage.

2.2.2 Decision Process

The level of concern for medical device software may be determined by sequentially answering the following five key questions, prior to mitigiation. If the answer to any one (or more) of the questions is yes, the software may be of major or moderate level concern. If the answer to any question is no, continue on to the next question. (Refer to Figure 2).

1. Does the device software control a life supporting or life sustaining device?

2. Does the device software control the delivery of potentially harmful energy which could result in death or serious injury, such as radiation treatment systems, defibrillators, and so forth?

3. Does the device software control treatment delivery, such that an error or malfunction with the delivery could result in death or serious injury?

4. Does the device software provide diagnostic information on which treatment or therapy is based, such that if misapplied it could result in serious injury or death?

   If yes, then the device may be of a "major" level of concern, especially in cases where diagnostic information would be misleading or inaccurate, or inappropriate therapy would be delivered. These situations may result in serious injury to the patient through an improper diagnosis of a serious medical condition or improper treatment.

   A device may provide data for which it is unlikely that the clinician will exercise independent judgement. The following are examples of scenarios that might arise:

   Major: a radiation treatment system delivering potentially harmful radiation therapy without user knowledge; and

   Moderate: a system providing incorrect diagnostic information which is readily and easily detected and overridden based on patient demographics, symptoms, and other tests.

   In the latter case, one in which clinical judgement would be exercised to override the information provided by a medical device (even in cases of incorrect data), the device would be considered one of lower level of concern.

5. Does the device software provide vital signs monitoring and alarms for potentially life threatening situations in which intervention is necessary?

If the answer is yes to any of the above questions, the following question should be answered:

   Does the software directly affect the patient so that failures or latent design flaws could result in death or serious injury to the patient, or does it indirectly affect the patient such that incorrect or delayed information could result in death or serious injury to the patient?

If the answer is yes, the level of concern is major. If the answer is no, then the following should be answered:

   Does the software directly affect the patient so that failures or latent design flaws could result in minor to moderate injury to the patient, or does it indirectly affect the patient such that incorrect or delayed information could result in non-serious injury to the patient?

If the answer is yes, the level of concern is moderate. If the answer is no, then the level of concern is minor.

In conclusion, reviewers should expect reasonably complete evidence to support the level of concern determination for medical device software.

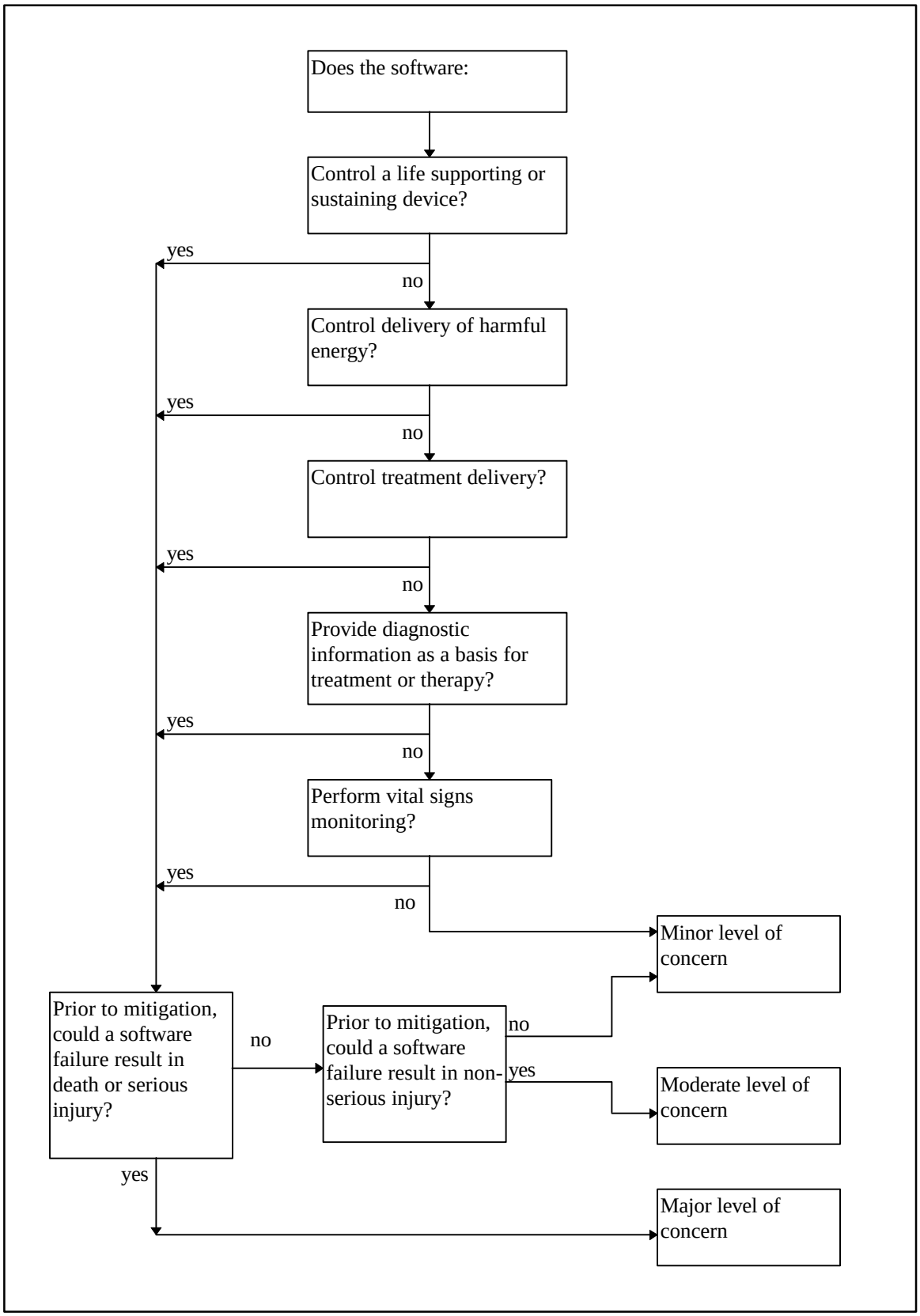

Figure 2: Determining Level of Concern

Section 3. Documentation in a Premarket Submission

Software documentation in premarket submissions should be consistent with the intended use of the device, level of concern of the software, and type of premarket submission. Table 1 summarizes the types of documentation that should generally be provided for a device in each of the three levels of concern. Note, if a device-specific guidance exists, it supersedes Table 1 Manufacturers may contact the corresponding ODE review division prior to making a submission to obtain any additional pertinent information.

The discussion that follows provides additional information about each of the software documentation categories described in Table 1. Note that depending on the level of concern of the software, it may not be necessary for the manufacturer to submit some or all of the documentation described for each category.

3.1 Level of Concern

Provide the level of concern of your software, and the supporting rationale (see Section 2 for details). If the level of concern is major, and you have not previously submitted a premarket submission for this type of device to ODE, it is strongly recommended that you contact the appropriate division prior to making the submission.

3.2 Software Description

Provide a comprehensive overview of the device features that are controlled by software, and describe the intended operational environment.

3.2.1 Device Features Controlled by Software

What is the role of the software in your medical devices? What does the software do, and as important, what does the software NOT do? How does the user interact with the software? What software functions can be controlled or modified by the user, and which are unchangeable? Which, if any, of the software features have hardware over-rides or backups?

This information should generally be provided in paragraph format, and should highlight major/significant software features. A detailed description of software requirements is addressed under subsection 3.5, Software Requirements Specification.

3.2.2 Operational Environment

The description of the intended operational environment should include the following:

- programming language

- hardware platform
- operating system (if applicable)*
- use of Off-the-Shelf components (if applicable)*

* If your device uses Off-the Shelf components, please refer to FDA's (draft) Guidance for Off-the-Shelf Software Use in Medical Devices (see Appendix C.6).

3.3 Device Hazard Analysis

Submissions for any level of concern should contain a device hazard analysis that takes into account all device hazards associated with its intended use, hardware, and software. The hazard analysis should include the following:

a) the hazardous event;
b) level of concern of the hazard;
c) the cause(s) of the hazard;
d) the method of control;
e) corrective measures taken, including aspects of the device design/requirements, that eliminate, reduce, or warn of a hazardous event, including a discussion of their appropriateness; and
f) testing to demonstrate that the method of control was implemented correctly.

This information is typically presented in a tabular format. When performing a hazard analysis, manufacturers should be careful to include all foreseeable hazards, including those resulting from intentional or inadvertent misuse of the device.

There are times when a fault tree analysis of the software is useful. For each identified hazard, a detailed analysis should be carried out to discover the conditions which might cause that hazard (see (c) above). There are various techniques, one of which is a fault tree analysis which involves identifying the undesired event and working backwards from that event to discover the possible causes of the hazard.

3.4 Software Requirements Specification (SRS)

The Software Requirements Specification documents the requirements for the software. This typically includes functional, performance, interface, design, and developmental requirements. Examples of some typical requirements that would be included in a SRS include:

a) hardware requirements, including microprocessors, memory devices, sensors, energy sources, safety features, communications, etc.;
b) programming language and program size(s);
c) interface requirements, including both communication between system components and communication with the user (e.g., printers, monitors, keyboard, mouse, etc.);
d) software performance and functional requirements, examples of which include:

- algorithms or control characteristics for therapy, diagnosis, monitoring, alarms, analysis, and interpretation (with full text references or supporting clinical data if necessary),
- device limitations due to software,
- internal software tests and checks,
- error and interrupt handling,
- fault detection, tolerance, and recovery characteristics,
- safety requirements,
- timing and memory requirements,
- identification of off-the-shelf software (if appropriate).

For a minor concern device, it is usually sufficient to provide only the functional requirements section from the SRS. For moderate and major concern devices, the complete SRS should be provided.

3.5 Architecture Design Chart

For minor level devices, submissions should contain a chart depicting the partitioning of the software into its functional subsystems. Note, it is not necessary to include each and every software module. The chart should depict the software architecture at the functional level.

For moderate and major concern devices, the chart should be more detailed, although still focused at the functional level. The submission should also include a list of functional modules, and a description of the role that each module plays in fulfilling the software requirements.

3.6 Design Specification

The software design specification is a description of what the program should do and how it should do it. It should both provide a high level summary of the design and specifications detailed enough such that a programmer is not required to make ad hoc design decisions. The software design specification should include documentation of the elements listed below, and will likely reference separate documentation for some of these elements:

- Software Requirements Specification, including predetermined criteria for acceptance of the program;
- Development standards and programming standards;
- Hazard Analysis;
- Systems documentation (context in which the program is intended to function, e.g. systems documentation narrative or context diagram);
- Hardware to be used;
- Parameters to be measured or recorded;
- Logic (logical structure, control logic, logical processing steps);
- Data Structures and data flow diagrams;
- Definitions of variables (control and data) and description of where they are used;

- Error and alarm messages;
- Supporting software (e.g. operating systems, drivers and other application software);
- Communications links (links among internal modules of the software, links with the supporting software and links with the hardware); and
- Security measures (both physical and logical security).

3.7 Traceability Analysis

Provide a traceability analysis or matrix which links requirements, design specifications, hazards, and validation. Traceability among these activities and documents is essential. This document acts as a map, providing the links necessary for determining where information is located. Although it is possible to document traceability through a shared organizational structure and common numbering scheme, appropriate information should be provided to guide the review through the information included in the submission.

3.8 Development

For moderate and major concern devices, sponsors should provide a summary (typically no more than 2-4 pages) of the software development life cycle plans. The summary should describe the processes that are in place to manage the various software development life cycle activities. Also for moderate concern devices, sponsors should provide a summary of the configuration management and maintenance plans.

In addition, for major concern devices, the submission should include an annotated list of the control/baseline documents generated during the software development process, as well as the configuration management and maintenance plan.

3.9 Validation, Verification and Testing

Software verification involves a systematic application of various analyses, evaluations, assurances, and testing of the software and its supporting documentation at each stage of the software development process, to assure that all the requirements specified for that stage have been fulfilled. Software validattion uses similar analytical techniques, but goes further to assure, to the extent possible, that the finished device (with its incorporated software) is appropriate for its intended use and will be reliable, and safe.

The following test information should be provided:

a) for a device in which the software is considered of minor level of concern:

- a software functional test plan with pass/fail criteria, data, and an analysis of the results;

b) for a device in which the software is considered of moderate level of concern:

- a description of the verification activities at the unit, integration and system level,
- a system level test protocol including pass/fail criteria, and test results;

c) for a device in which the software is considered of major level of concern:

- a description of the verification activities at the unit, integration and system level,
- unit, integration and system level test protocols including pass/fail criteria, test report, summary, and test results.

All testing information should include the version and revision identifiers for the software and a discussion of testing results should include a discussion of how the following was tested (when applicable):

- fault, alarm, and hazard testing,
- error, range checking, and boundary value testing,
- timing analysis and testing,
- special algorithms and interpretation tests and analysis,
- stress testing,
- device options, accessories, and configurations testing,
- communications testing,
- memory utilization testing,
- qualification of off-the-shelf software (see Appendix C.6),
- acceptance and beta site testing, and
- regression testing.

3.10 Revision Level History

For moderate and major concern software, the submission should include the revision history log, documenting all major changes to the software during its development cycle.

3.11 Unresolved Anomalies (Bugs)

For moderate and major concern software, the submission should include a list of all unresolved software anomalies. For each anomaly, indicate the problem, the impact on device performance, and, if appropriate, any plans or timeframes for correcting the problem. This list of bugs should be communicated to the user in the device labeling.

3.12 Release Version Number

For all levels of concern, the submission should include the release version number and date for the software that will be included in the marketed device.

SECTION NUMBER	SOFTWARE DOCUMENTATION	MINOR CONCERN	MODERATE CONCERN	MAJOR CONCERN
2, 3.1	Level of Concern	All levels of concern		
3.2	Software Description	All levels of concern		
3.3, 4.3	Device Hazard Analysis	All levels of concern		
3.4, 4.2	Software Requirements Specification (SRS)	Software functional requirements from SRS	SRS	
3.5	Architecture Design Chart	A chart depicting the partitioning of the software system into functional subsystems	A chart depicting the partitioning of the software system into functional subsystems, listing of the functional modules and a description of how each fulfills the requirements.	
3.6	Design Specification	No documentation is necessary in the submission.	Software design specification document	
3.7	Traceability Analysis	No documentation is necessary in the submission.	Traceability among requirements, identified hazards, and Verification and Validation testing.	
3.8, 4.1	Development	No documentation is necessary in the submission.	Summary of software life cycle development plan, including a summary of the configuration management and maintenance activities.	Summary of software life cycle development plan. Annotated list of control documents generated during development process. Include the configuration management and maintenance plan documents.
3.9	Validation, Verification and Testing (VV&T)	Software functional test plan, pass / fail criteria, and results	Description of VV&T activities at the unit, integration and system level. System level test protocol including pass/fail criteria, and tests results.	Description of VV&T activities at the unit, integration and system level. Unit, integration and system level test protocols including pass/fail criteria, test report, summary, and tests results.
3.10	Revision Level History	No documentation is necessary in the submission.	Revision history log	
3.11	Unresolved anomalies (bugs)	No documentation is necessary in the submission.	List of errors and bugs which remain in the device and an explanation how they were determined to not impact safety or effectiveness, including operator usage and human factors.	
3.12	Release Version Number	Version number and date for all levels of concern.		

Table 1 Documentation in a Premarket Submission

Section 4. Risk Management Activities During the Software Life Cycle

This section provides an overview of risk management activities and where they fit into the medical device software life cycle. The software life cycle is briefly described. For more specific details of life cycle activities, refer to FDA guidance on "General Principles of Software Validation."

4.1 Life Cycle Models and Development Methodologies

The software life cycle is a microcosm of the entire device life cycle. The manufacturer can choose a software life cycle model and development methodology that is appropriate for their device and their organization. Generally, the life cycle model selected should include activities for risk management, requirements analysis and specification, design (both top level and detailed), implementation (coding), integration, validation, and maintenance. A software life cycle model should be understandable, thoroughly documented, results oriented, auditable, and traceable, and should promote appropriate feedback within the development process. "Code-and-fix" is a commonly used, but not very effective approach to the software life cycle, because it provides no means for identifying risks, assessing quality, and identifying and eliminating anomalies early in the development process.

There are a variety of life cycle models, such as: waterfall, spiral, evolutionary, incremental, top-down functional decomposition (or stepwise refinement), formal transformation, etc. Medical device software may be produced using any of these or other models, as long as adequate risk management activities and feedback processes are incorporated into the model selected. It is feasible to intermix different life cycle models and methodologies among subsystems and subcomponents (i.e., hardware, software, materials, etc.). Terminology from model to model and methodology to methodology may vary. A generic life cycle model is depicted in Figure 3, depicting the relationship of risk management activities to life cycle activities.

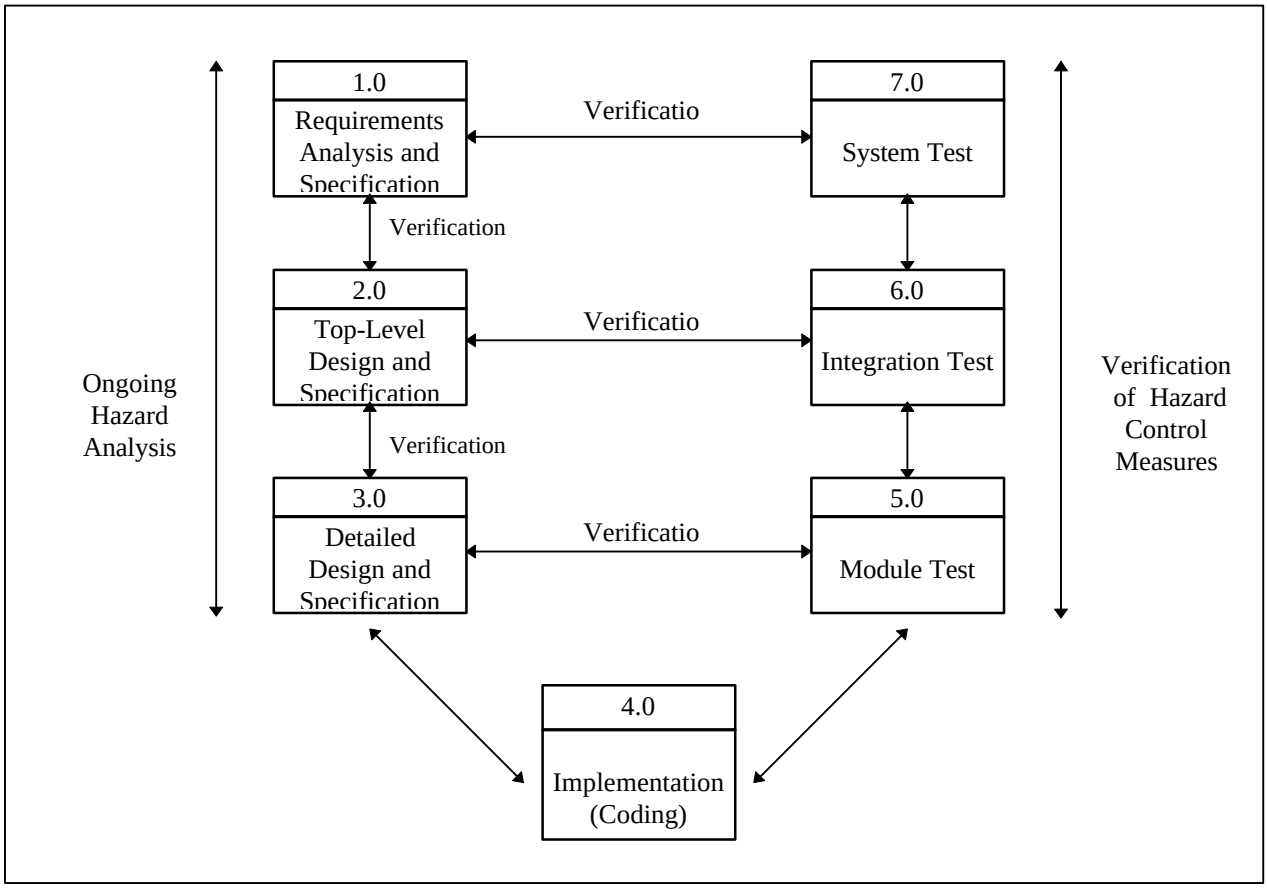

Figure 3: Generic Software Life Cycle Model

4.2 Requirements Analysis and Specification

Early in the device design process, there is an activity which identifies and analyzes customer or end-user functional and performance requirements for the device, and determines which of those device (system) requirements will be allocated to software. It is important to define the role of the software in the device at this time, in particular with regard to risk-related functions. During this requirements activity, the functions to be performed, controlled, or monitored by the software are documented. Software quality characteristics, such as human factors, functional characteristics, response times, output, safety requirements, etc., are defined, along with acceptance criteria.

Software safety requirements are derived from the preliminary hazard analysis and ongoing risk management activities, as requirements are updated throughout the life cycle process. Any hazardous software functions are identified, evaluated, and traced to be able to show subsequent mitigation of the risk associated with those functions to an acceptable level.

4.3 Risk Management

Risk management is a combination of risk analysis and risk control activities. Risk analysis is used to estimate the risk associated with the use of the device, and risk control is used to mitigate the risk to an acceptable level. It is normal for a manufacturer to have a risk management plan that is used to define and control the risk analysis and risk control activities. These activities are ongoing throughout the device life cycle. Risk analyses should be performed for the device as an entity, as well as major components/subsystems. Appropriate techniques should be chosen so that risk analyses for the software, electronics, biomaterials and so forth, can be effectively integrated and analyzed at both the device level and at the subsystem level.

Several national and international consensus standards, such as those cited in Appendix B, can assist manufacturers during this process.

There are many dimensions to risk, for example: cost, integrity, security, safety, etc. For FDA's review of medical devices, the primary focus is on the safety dimension of risk. Software risk management is a subset of the overall risk management of the device. Risk is reduced when the severity of the consequence of the hazard is reduced and/or the likelihood of the hazard occurrence is reduced. When a hazard is specifically linked to the software, the likelihood of hazard occurrence is directly related to the failure rate of the software. The calculation of accurate software failure rates is difficult, if not impossible. Therefore, risk reduction and mitigation techniques for software should be employed to control the severity of a hazard assuming that the likelihood of occurrence is unacceptably high.

4.3.1 Risk Analysis Activities

Risk analysis includes hazard analysis and risk estimation. Hazard analysis provides documented identification for potential device hazards. Potential device hazards are identified for all reasonably foreseeable circumstances. Hazard analysis is conducted in accordance with established procedures. It should begin early in the life cycle, as requirements are established, and should be updated as development progresses, to assess the effectiveness of hazard mitigation and whether any new hazards have been introduced. Device hazards should be considered for their effect on the following: patients, operators, bystanders, service personnel, and the environment.

For each identified hazard, a list of possible initiating causes is developed. Initiating causes can come from any of the following areas: human factors, hardware faults, software faults, integration errors, and environmental conditions. A variety of methods can be used to perform the hazard analysis (e.g. fault tree analysis, failure modes and effects analysis). In general, different methods will be used during different phases of the development life cycle as more becomes known about the end product. The manufacturer is expected to select the appropriate methods for their device and its intended use. For purposes of this document, a software related hazard is defined as any device hazard that has its initiating cause or a major contributing cause in software. For software related hazards, the hazard analysis documentation should identify traceability from the device level down to the specific cause in the software.

Risk estimation is conducted for each identified hazard. The risk estimation is based on the

severity of resulting consequences and the likelihood of occurrence. For each hazard a severity level should be assigned. Severity levels can be defined using the level of concern terminology (major, moderate, and minor) used in this document.

For each hazard, a likelihood of occurrence should be assigned. In the case of software related hazards, one component of this likelihood is directly related to the software failure rate. The software failure rate is the result of systematic (versus random) software faults. Due to the nature of systematic faults, the accurate estimation of software failure rates is difficult. Unless the accuracy of the software failure rate can be confirmed it will not be appropriate to control risk based on estimated software failure rate. For software related hazards, software failure rates need not be calculated if the manufacturer assumes that the software failure rate is at an unacceptable level. Using this approach, the manufacturer will be able to concentrate resources on creating design solutions that reduce and/or eliminate the severity of hazards.

A device may have multiple potential hazards associated with it. Likewise, each hazard may have multiple potential causes. The goal should be to identify all potential causes for each hazard. The degree of effort and detail in characterizing potential causes of a hazard should be commensurate with the severity of resulting consequences. The methods used to identify hazards and their causes, and to categorize severity should be documented.

4.3.2 Risk Control Activities

Risk control is intended to eliminate a hazard or to mitigate the estimated risk to an acceptable level. Risk control methods are directed at the cause of the hazard. Several risk control methods and their priority are as follows:

1) eliminate or reduce the risk by inherent safe design or redesign;

2) reduce the risk by protective measures; and

3) reduce the risk by adequate user information, such as sufficient warnings.

For each identified hazard, the manufacturer should identify the risk control method that was used to eliminate the risk or reduce the risk to an acceptable level. For each software related hazard, the manufacturer should indicate the severity level after the risk control method has been implemented. The goal is to reduce all software related hazards to a minor level of concern.

A determination is made about the appropriateness of the residual risk for each hazard/cause combination. Following this, a determination is made as to whether or not the risk control measures introduced any new hazards. If so, the process is repeated. Referring to Figure 4, steps 2 through 7 are repeated for each potential hazard while steps 4 through 6 are repeated for each potential cause. After all potential hazards have been evaluated, a final determination is made about the device safety.

4.3.3 Documentation

Several work products result from the ongoing risk management process. A hazard analysis by itself is not sufficient. Manufacturers should also document:

1) a description of the identified hazard;

2) the severity level of the hazard (major, moderate, or minor);

3) the specific software cause of this hazard, so it can be traced to the specific location in the software;

4) risk control method employed;

5) test or verification method used to confirm the risk method employed; and

6) severity level of the hazard after the risk control method has been implemented.

- what the estimated severity of each hazard is and how it was categorized;
- what risk reduction and mitigation techniques were implemented and how their effectiveness was assessed; and
- testing and evaluation demonstrating the implementation of the safety features.

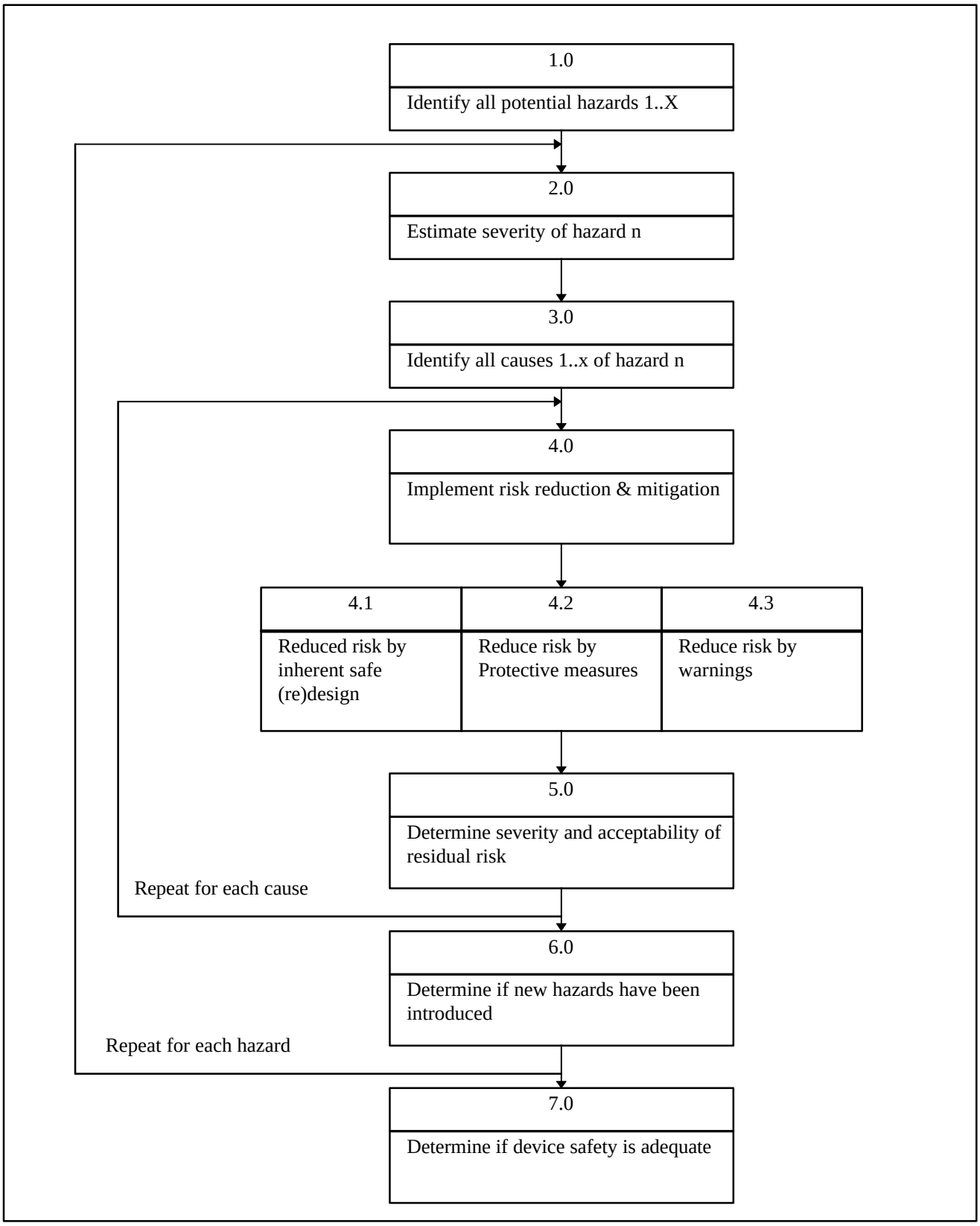

Figure 4: Risk Management Process (Adapted from IEC 60601-1-4)

Appendix A. Technological Issues and Special Topics

This appendix, which is informative, provides a brief discussion of a variety of special topics that relate to software and premarket submissions. These topics are mentioned to give reviewers a "heads-up" to alert them that the software may incorporate these features and may need special attention or further research. For a more comprehensive discussion of individual topics, consult the references in Appendices D and E.

A.1 Artificial Intelligence, Expert Systems, and Neural Networks

Artificial intelligence (AI) is a field of research in computer science which studies methods and techniques by which computational machinery may exhibit behavior and responses similar to those exhibited by humans and other biological organisms. Two areas where concepts derived from artificial intelligence research have been applied to problem solving tasks are expert systems and artificial neural networks.

Expert systems attempt to model very specific areas of human knowledge or expertise by distilling the experience of human experts into a set of algorithms which can be executed by software. The expert system often consists of a knowledge base (consisting of rules, heuristics, or relationships between objects or data) and an inference engine which manipulates the knowledge according to selected criteria. Expert systems use these rules or heuristics on facts input into the system to solve problems in a narrow, well-defined domain or area. Typical applications for expert systems include circuit analysis and design, fault detection and diagnosis, automated finance assessment and loan processing, and medical diagnosis and therapy recommendation.

An artificial neural network (ANN) is a data processing architecture that is modeled on principles exhibited in natural or biological neural networks. Artificial neural networks are often used to represent or process nonlinear functions applied to large data sets. Artificial neural network engines can be implemented in software, hardware (using parallel processing architectures) or a combination of both. Artificial neural networks are well-suited for detecting trends or patterns in data, and are typically used for speech and natural language processing, machine vision and image recognition, financial trend forecasting, and automated medical image processing. Artificial neural networks are represented symbolically as an interconnected network of nodes arranged in a specific topology or configuration. Links between nodes represent dependencies between nodes and have weights associated with each link representing the strengths of the dependencies. Artificial neural networks typically have an input layer, hidden or processing layers, and an output layer. The links between nodes, and potentially the topology of the network itself, are adjusted for specific tasks by training of the network, which involves exposing the network to representative data sets to be processed. Output from the network are compared to desired results and corresponding adjustments are made to reduce any discrepancies between the desired output and the actual output. The field of artificial neural networks is a rapidly expanding one, and many artificial neural network models, learning methods, topologies, and training regimens currently exist with others being created constantly.

Expert systems and artificial neural networks are relatively new technologies which are increasingly being incorporated into medical devices. However, they pose special challenges

regarding the validation of the core processing architectures: the knowledge base and inference engine for expert systems, and the neural net engine for artificial neural network systems.

The knowledge base of expert systems needs to be verified for accuracy of information and of the relationships between data objects or object classes. The heuristics and rules governing the inference engine need to be analyzed to ensure that there are no logical or common-sense contradictions or paradoxes that exist or that are possible by the system in operation. Any output or determination produced by an expert system should also be accompanied by the reasoning path followed by the software to reach its conclusions. Such information should serve to allow the user to determine if the reasoning path followed by the expert system is sound, or if other valid reasoning paths were not appropriately explored.

Artificial neural networks are, by their very nature, difficult (if not impossible) to qualify using traditional software engineering methodology. The strength of artificial neural networks, the ability of the network to "learn" by example and self-adjust its internal parameters or configuration, is what makes validation of artificial neural network engines problematic. The performance and behavior of artificial neural networks are determined by selective exposure to training sets and its environment, not by strict specifications. In some cases, artificial neural networks can behave in a non-deterministic manner (that is, the same input may produce different outputs at different times). Traditional software engineering methodologies are designed and intended for deterministic software implementations.

The design, assumptions, learning method, and training set data for an artificial neural network should be evaluated for appropriateness and correctness. The network designers should justify and explain the choices made for the artificial neural network model, topology, and training sets, as well as explain and justify the data set class that the artificial neural network is intended to analyze or process. The designers should describe how overfitting or overtraining of the network was avoided, i.e., when it was decided that the network was "trained" sufficiently to enable appropriate performance before the network begins to extract irrelevant details from the data from overexposure to example sets. When examining the training set presented to the neural network, it is important to ensure that the features to be extracted (such as a specific pattern to be detected) remain the common element within the training set data. Once training has been completed, additional data sets should be processed through the network to ensure that the performance remains as expected and relevant data is extracted appropriately. Tests should be performed to ensure that the network was not trained to detect a particular peculiarity of the training set instead of the intended features. Raw data processed by the system should be presented to the user for comparison with the output from the system.

A.2 Automatic Code Generators

Some computer assisted software engineering (CASE) tools (see B.3) include automatic code generators. Software from automatic code generators should be validated the same way as any other software. This may include, where appropriate, requirements traceability and code walkthroughs.

A.3 Computer Assisted Software Engineering (CASE) Tools

Computer assisted software engineering (CASE) tools are often used to automate or assist in software development and increase productivity. There are many different tools and types of tools available. At present there is not a single tool or suite of tools from a single vendor which covers all phases of the development life cycle. This raises concerns about the accuracy of outputs from CASE tools; particularly when developers work in a non-integrated or multi-vendor tool environment.

A.4 Changes and Modifications

Changes and modifications will occur during the development life cycle, after a device is fielded, and as a product line matures. Modifications may take the form of requirements changes, design changes, corrections, or enhancements. The extent and nature of the modifications will determine whether: (1) they can be accommodated by configuration management and change control procedures; or (2) the requirements of the entire development life cycle apply. Of primary concern is the effect of the modifications on risk analysis and control measures.

Examples of changes include:

1) New hardware platform. This could be migrating to a newer version of the same architecture family, such as i386 to i486, or changing architectures such as from i486 to a workstation.

2) New operating system. This could be migrating to a newer version of the same operating system or changing operating systems.

3) New compiler. This could be migrating to a newer version of the same compiler or changing compilers.

4) New functionality. This includes new features and capabilities that are provided for the end-user.

5) Design enhancements and corrections. This includes changes to the internal software design that may or may not be visible to the end-user. These changes are undertaken to improve software performance, safety, and reliability.

The extent and nature of the changes and modifications will also determine whether a new 510(k) submission is required or information about the changes and their effect on software safety and reliability may be submitted. Refer to the latest version of FDA/CDRH publication "Deciding When to Submit a 510(k) for a Change to an Existing Device." Changes to an investigational device or PMA device should be handled consistently with those types of devices and submissions. (See Appendix C.6.) For more detailed information regarding software changes, refer to the medical device Quality System regulations (see Appendix C.6) and General Principles of Software Validation (see Appendix C.6).

A.5 Clinical Data

When new algorithms are employed, whether for treatment, diagnosis, or monitoring, clinical data

may be necessary to establish the modified device's safety and effectiveness. This does not imply that every new algorithm needs to be clinically tested, especially since new algorithms may be developed for issues that do not relate to the inherent risk of a device, such as a new communication protocol for an internal printer. Manufacturers, however, should be aware that utilizing new algorithms for various aspects of treatment, diagnosis, interpretation, monitoring, etc. may need to be clinically validated by appropriate clinical trials that yield relevant clinical data and results. The reviewing division within CDRH should also be contacted as early as possible for proper guidance and requirements. Determinations of whether a particular device is a significant risk device and requires an IDE submission to the agency should be made by the reviewing division and/or an institutional review board (IRB). Regardless of whether an IDE application is submitted, adequate informed consent should be made available to the patient and the study should be approved by an IRB.

A.6 Closed Loop and Target Control

Closed loop systems typically include patient feedback, while target control typically "estimates" patient response. In either case, control of a device is based on 'real' or 'estimated' patient data. Typically, these types of devices have required clinical data to support the algorithms on which they are based. In either case, the design and architecture of the system (including software) should allow for partitioning of the system so that complexity is reduced, and safety and testability are maximized. Safety is a critical issue since the clinician is removed from direct control of the device. Adequate risk assessment and mitigation activities should be performed during the software life cycle process, and failure analyses techniques should include assessing multiple event failures. Some single event failures may not pose a direct safety hazard; however, this may change when multiple event failures occur concurrently or in a particular sequence. Even if a legally marketed device is incorporated into a closed loop or target control system, requalification of the device in order to assess its appropriateness for incorporation into one of these kind of systems may be necessary, especially since the behavior in a closed loop system may alter the way a device typically functions. Due to the nature of using a device to either monitor or control therapy in a closed loop system, qualification of the device and any modifications should also be a part of the software life cycle processes and methodologies.

A.7 Custom Operating Systems

Real-time systems may utilize a custom operating system which is designed and developed for a particular use, especially on a higher level of concern device. An executive system is essentially an operating system much like that on a personal computer that manages processes and resource allocation. They typically include a clock, an interrupt handler, a scheduler, a resource manager, a dispatcher, a configuration manager, and a fault manager. Monitoring and control systems are real time systems which are designed to a generic architecture and are used for checking sensors which provide information about the system's environment and take actions depending on the sensors reading. Monitoring systems take action when some exceptional sensor value is detected. Control systems continuously control hardware actuators depending on the value of associated sensors. Another type of real-time system is a data acquisition system which collect data from sensors for subsequent processing and analysis. These also typically have a generic architectural design.

A.8 Data Compression

Many data processing devices, such as a holter monitor, involve storage of large volumes of data. To reduce the storage requirements, there is a need to reduce the redundancy in the data representation. That is, to compress the data. The compression and expansion of data can be implemented in hardware, firmware, or software. If compression and expansion is done in software, there is an increase in complexity in the software and there are many techniques that can be used to accomplish this. Data compression can be divided into two categories: irreversible and reversible. Irreversible techniques involve a reduction of the physical representation of the data, usually referred to as data compaction. All information is considered to be relevant in data compression, and the compression will be followed later by expansion which recovers the original data. It is the manufacturer's responsibility to show that the expanded data provides an accurate recount of the original data.

A.9 Embedded and Real-Time Systems

Embedded and real-time systems include embedded software, software using a real-time operating system, programmable logic arrays (PLAs), programmable logic devices (PLDs), etc. This type of software poses unique concerns about safety and reliability because, in general, the development environment is different than the intended operational environment. Techniques and/or simulators and emulators should be employed to analyze the timing of critical events and identify non-deterministic conditions.

A.10 Human Factors and Software Design

The focus of human factors is user interface design. Poor design induces errors and inefficiency among even the best-trained users, especially under conditions of stress, time constraints, and/or fatigue. Although labeling (e.g., user documentation) is extremely important to good performance, even well-written instructions are cumbersome to use in tandem with actual operation. Also, it's difficult to write coherent documentation which describes awkward operating procedures.

Although both hardware and software design influence the user's performance, the logical and informational characteristics provided via software are increasingly crucial. Data presented in an ambiguous, difficult-to-read, or counter-intuitive manner poses the threat of an incorrect reading, misinterpretation, and/or improper data entry. An example might be a crowded display with cryptic identifiers combined with a time lag between user response and displayed feedback. Such design characteristics overtax the user's abilities (e.g., memory, visual perception, decision-making, etc.), and resultant errors may have serious consequences. For more detailed information regarding human factors, refer to Do It By Design (see Appendix C.6).

A.10.1 Common Problems

The logic and simplicity of control-activated operations and information access/manipulation is crucial, no matter what the program medium. Below are problem areas which lead to errors, and most are generally applicable to devices regardless of manner of control operation and

information display or feedback:

- Uncertain or no feedback following input;
- missing or ambiguous prompts;
- automatic resets or defaults not initialized by the user;
- unreasonable mental calculations required;
- no query for critical input;
- complex command structure;
- unfamiliar language/coding/acronyms, mnemonics, etc.;
- inconsistencies among formats for successive or co-located displays;
- conventions (e.g., color) contradictory to user stereotypes/expectations;
- ambiguous symbols or icons;
- no appropriate lock-outs or interlocks;
- illogical or cumbersome control sequences or screen call-up ("navigation");
- no status information; etc.

A.10.2 Examples

Many user errors induced by software design are attributed to other factors due to the fact that often little, if any, physical evidence remains after the fact. Also, software-related errors can be subtle. For example, confusion from illogical data entry sequences can induce errors only indirectly related to these procedures. In any case, there are many software examples gathered from incident files, recalls, and analytic findings.
Below are a few examples:

a. In at least one radiation device there have been problems due to the fact that user failure to input a dosage (time or amount) leads to a default value. The user was not queried; nor was the default value displayed or a warning/alarm presented.

b. A neonatal monitor didn't alarm for very high heart rate. It switches to "Half-Rate" display when rate is over 240 BPM. The patient, an infant, was hypoxic and required emergency treatment.

c. CDRH discovered that a clinical batch analyzer clears all patient information fields when

the operator attempts to remove any incorrect information for that patient. Also, "cleared" values are reassigned in such a way to increase the number of false negative readings over the batch.

d. There have been numerous recalls of devices in which slight deviations from prescribed operating sequences will disable the device, in some cases without any feedback to the operator.

A.10.3 Proper Analysis and Testing Pays Off

Good human factors design involves the following; a) integrating users into the design process early; b) close coordination of software and hardware efforts; c) including user "advocates" and subject matter experts on the design team; and d) performing iterative analyses, simulations, and usability tests. Tools may involve surveys, focus tests, interviews, storyboards, documentation, etc.

The human factors engineering process can elucidate subtleties that even user-oriented designers can overlook. For example, symbols, icons, colors, abbreviations, etc. can convey a great deal of information reliably, economically, and quickly; but a priori assumptions about their meaning and clarity can be incorrect, depending upon variability among user populations, work settings, device experience, and conventions outside of the medical area. Analysis, testing and the judicious use of guidelines and standards can be incisive. In general, the human factors payoff includes fewer injuries or deaths, reduced training costs, and more marketable products. A full-length primer on human factors considerations for medical devices, titled "Do it by Design", is available (see Appendix C.6).

A.11 Off-the-Shelf (OTS) Software

Please refer to the Guidance for Off-the-shelf Software Use in Medical Devices (see Appendix C.6).

A.12 Open Systems and Open Systems Architecture

Open systems may be viewed differently by many people. One view is the ability to enable dissimilar computers to exchange information and run each other’s software via interfaces from independent vendors. These would be considered “open” operating systems with increased interoperability, flexibility, and portability. The idea is freeing proprietary pathways within each system. Another view is one which is used more frequently and pertains to sharing device control and communications within networks, across devices, etc. This allows for flexibility in configuring networks and systems, and may include various aspects of medical devices and hospital information systems such as intensive care units, critical care units, operating rooms, pulmonary and cardiac labs, clinical laboratories, radiology laboratories, etc. This sharing of information, control, and network time and space increases the complexity of medical devices. This may not be desirable for higher risk devices, especially since the environment will be difficult to model during verification and validation. Appropriate test suites and test cases may be difficult to analyze from a “completeness” point of view; determining when enough testing has taken place

and test completion criteria has been met.

The term “open system” has typically referred to the flexibility in using several different vendor devices in a network of some kind, which would require that each vendor have appropriate knowledge of proprietary device drivers for appropriate communication. During the verification and validation process, all information needed for proper communication should be well known by all so that devices can be properly developed and tested. Because medical devices of “higher” level of concern require a more robust operating environment, open systems may not be the most appropriate approach.

A.13 Process Control Software

Process control software is a Good Manufacturing Process (GMP), Good Laboratory Practice (GLP), or Good Clinical Practice (GCP) issue. While the same development life cycle and risk management activities apply, the primary concern is that the software works correctly in the intended manufacturing, laboratory, or clinical process.

A.14 Redundant Displays

Redundant displays, even if secondary, are relied upon as much as the original device monitor. A redundant display allows information to be displayed at a remote location (or different position from the parent device) and sometimes allows for limited control of the device. Manufacturers should be aware that redundant displays are considered medical devices, just as the parent device, and are reviewed as such. Therefore, software development activities for such a device should be treated with the same regard. This is especially true if the device is part of a “higher” level of concern monitoring system.

A.15 Research Shareware/Freely Distributed Software

Research shareware is software that: 1) is developed in a university/research setting; 2) receives limited distribution in order to obtain feedback from beta testing; and 3) is developed and distributed with no intent to market. Research shareware may be distributed in the form of object code, source code, and/or source code listings. The functionality, safety features and procedures, and reliability should be validated for the intended use. Should research shareware be incorporated into a commercial product, the end manufacturer is responsible for validation, verification, and support activities. The CDRH Office of Compliance issued a letter, dated October 20, 1995, concerning regulatory responsibilities for research shareware.

A.16 Reuse and Libraries

"Software reuse involves reusing existing components rather than developing them specially for an application. Systematic reuse can improve reliability, reduce management risk, and reduce development costs. Software development with reuse needs a library of reusable components that can be understood by the reuser; information on how to reuse the components should also be provided. Systematic reuse requires a properly catalogued and documented base of reusable components. Reusable software components do not simply emerge as a by-product of software development. Extra effort should be added to generalize the system components to make them

reusable. Abstract data types and objects are effective encapsulators of reusable components. Development according to standards for languages, operating systems, networking and graphical user interfaces minimizes the costs of implementing a system on different types of computers." (Sommerville 1996, see Appendix E.1.)

If specifications, design documentation, test plans and procedures are written for the lowest level software components, they can be reused also. Prior to reusing software, an evaluation should be made of the appropriateness of the intended use in the new application and its affect on safety and reliability. If the development environment or life cycle processes differ when software is reused and supporting documentation is not consistent with the new software life cycle methodologies, some re-engineering may need to occur to import the reused software into the new environment.

A.17 Security and Privacy

Security can be viewed in many ways. Preventing access to data or records is one view. It may also pertain to accidental or intentional data manipulation, corruption or destruction that may occur through environmental factors such as a power failure which causes data to be lost. Software should be designed, verified, and validated so that these accidental, intentional, and/or environmental data losses do not occur.

Security may also be viewed as only allowing access to records by authorized parties. For example, records that are maintained regarding anesthesia delivery and monitoring in the operating room or ICU devices that maintain patient records should not be accessible to everyone. When records are modified, an indication that the data or record was modified should appear in the record and be printed on the patient report so that it is known that someone modified the data that was recorded and retrieved by the device. Not allowing for such a modification could be a potential solution. However, with many different types of editors and data conversion programs, this is virtually impossible to assure unless proprietary encryption is used so that records would not be readable by other devices, programs, or computers. And there are some devices where data manipulation may be desirable if the user does not agree with an event marker or interpretation offered by a medical device, and may need the opportunity to override the decision on the record. It is not within the scope of this document to provide solutions to computer/software/data security issues. It is, however, in the scope of this document to raise this issue and ask that manufacturers consider this during the development of software. Some common solutions would be to provide software and data backup on a regular basis, password protection, data recovery methods, and utilize well designed and tested encryption/decryption algorithms when appropriate.

A.18 Stand-Alone Software

A draft policy is being developed concerning the regulation of stand-alone medical software products.

A.19 Software Accessories

A software accessory is a software unit which is intended to be attached to or used in conjunction with another finished device, although an accessory may be sold or promoted as an independent

unit. Software accessories to medical devices are typically programmed calculators or software which:

a. accept data from the user and modify it for input to a medical device, or

b. take data from a medical device and modify it for presentation to the user.

Some examples of software accessories are:

Alphafetaprotein (AFP) calculator,
Radiation therapy treatment planning software,
Intraocular lens power calculator,
Digital imaging protocol conversion software,
Pacemaker rate response factor calculator,
EEG and ECG waveform analysis software
Hemodialysis calculator
Drug library editors for infusion pumps

By law, any product that is a component of or an accessory to a medical device, is itself also a medical device. FDA policy is that unless separately classified, a component, parts or accessory to a medical device is regulated in the same way as its parent device.

A.20 User Modifiable Software

User modifiable software includes situations where the user is able to configure a menu, the display screen, alarm and performance limits, as well as select normal values, data base information, or input their own interpretations, normal values and text. This is also a human factors issue (see A.10), but the issue of proper verification and validation during the software life cycle becomes difficult since users can do virtually anything during use of a device. Beta site testing is important during software validation since it allows the users to use and try to intentionally break the system to ensure that proper safeguards have been incorporated. This kind of testing may be hard to duplicate off site unless users are invited to a facility during development to better facilitate the verification and validation processes. Employing appropriate requirements, device limitations, and design constraints for user modifiable software is a vital human factors and safety concern. Life cycle processes, including testing and analysis, should account for these concerns.

A.21 Year 2000

Many computer systems and software applications currently in use may experience problems beginning January 1, 2000, due to their use of two-digit fields for date representation. Computer systems may not be able to accurately perform computations involving the year 2000 using only the last two digits, “00”. This may adversely affect the correct functioning of medical devices and/or their manufacturing and quality control processes.

To ensure safety and efficacy of medical devices, manufacturers should initiate appropriate actions to avoid any potential problems. CDRH recommends the following:

1. For future medical device applications, manufacturers should demonstrate that software designs for medical devices and their manufacturing and quality control processes can perform accurate date recording and computations in the year 2000 and beyond.

2. For currently manufactured medical devices, manufacturers should conduct a hazard or safety analysis to determine whether device safety or effectiveness could be adversely affected be the year 2000 date change.

APPENDIX B. Relevant National and International Consensus Standards

The following list is a collection of current voluntary national and international consensus standards that are directly or indirectly related to medical device software safety, reliability, and life cycle issues. The list is not exhaustive and is provided for consideration. It is understood that standards are continually undergoing update/reaffirmation cycles; accordingly the newest approved version should be used. The selection of a particular standard or set of standards will depend on many factors, including the design/development methodology, the type of device, the type of software, and standard corporate practices.

The standards are grouped by subject areas and may be obtained from the American National Standards Institute (ANSI) or the Institute of Electrical and Electronics Engineers, Inc. (IEEE) at the addresses below. Two special volumes should be noted: (1) ANSI publishes a complete volume of all of the ISO 9000 compendium standards; and (2) IEEE publishes a complete volume of all of their software engineering standards.

ANSI
11 West 42nd Street
New York, NY 10036
212.302.1286 (fax)
212.642.4900 (voice)

IEEE
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
908.562.1571 (fax)
908.562.3811 (voice)

B.1 General Life Cycle Activities

1. ANSI/IEEE 1058 Standard for Software Project Management Plans.

2. ANSI/IEEE 610.12 Software Engineering Terminology.

3. ANSI/IEEE 1063 Standard for Software User Documentation.

4. ANSI/NISO Z39.67 Software Description.

5. ANSI/IEEE 1002 Standard Taxonomy for Software Engineering Standards.

6. ANSI/IEEE 1074 Standard for Developing Software Life Cycle Processes.

7. ANSI/IEEE 1016 Recommended Practice for Software Design Descriptions.

8. ANSI/IEEE 1016.1 Guide for Software Design Descriptions.

9. ANSI/IEEE 1045 Standard for Software Productivity Metrics.

10. ANSI/IEEE 830 Recommended Practice for Software Requirements Specifications.

11. ANSI/IEEE 1028 Standard for Software Reviews & Audits.

12. ANSI/IEEE 1062 Recommended Practice for Software Acquisition.

13. ANSI/IEEE 1220 Trial-use Standard for the Application and Management of the System Engineering Process.

14. ISO/IEC 12207 Information Technology -- Software Life Cycle Processes.

B.2 Safety and Reliability

1. Medical Electrical Equipment - Part 1: General Requirements for Safety - 4. Collateral Standard: Programmable Electrical Medical Systems, IEC (DIS) 60601-1-4.

2. ANSI/IEEE 982.1 Standard Dictionary of Measures to Produce Reliable Software.

3. ANSI/IEEE 982.2 Guide for the use of Standard Dictionary of Measures to Produce Reliable Software.

4. ANSI/IEEE 1012 Standard for Software Validation & Verification.

5. ANSI/ANS 10.4 Nuclear Computer Programs.

6. ANSI/AIAA R-013 Software Reliability.

7. ANSI/IEEE 1228 Standard for Software Safety Plans.

8. Developing safe, effective, and reliable medical software, 1991 AAMI Monograph (MDS-175). [available from AAMI, 3330 Washington Blvd., Suite 400, Arlington, VA 22201-4598, 703.276.0793 (fax), 703.525.4890 (voice)]

9. IEC 1508 Functional safety: safety related systems
Part 1: General requirements;
Part 2: Requirements for programmable electrical systems (PES);
Part 3: Software requirements;

Part 4: Definitions and abbreviations of terms;
Part 5: Guidelines for the application of Part 1
Part 6: Guidelines for the application of Parts 2 and 3;
Part 7: Bibliography of techniques and measures.

10. ISO/IEC JTC1/SC7 WG9 Project 7.30 Software Integrity Levels (working draft 1.0.

11. (committee draft) IEC TC 56(secretariat)410 Dependability - Risk analysis of technological systems.

12. (committee draft) IEC SC 45A(secretariat) Control systems important to safety - 1st supplement to IEC 880.

13. IEC 812: Analysis techniques for system reliability - Procedure for failure mode and effects analysis (FMEA).

14. IEC 1025: Fault tree analysis (FTA).

B.3 Quality Assurance

1. ISO 8402 Quality Management and Quality Assurance Vocabulary

2. ISO (DIS) 8402/DAM 2 Quality Management and Quality Assurance Vocabulary Amendment 2

3. ISO 9000: Quality Management and Quality Assurance Standards - Guidelines for Selection and Use

4. ISO (DIS) 9000-1 Quality Management and Quality Assurance Standards - Part 1: Guidelines for Selection and Use

5. ISO (DIS) 9000-2 Quality Management and Quality Assurance Standards - Part 2: Generic Guidelines for the Application of ISO 9001, ISO 9002, and ISO 9003

6. ISO 9000-3 Quality Management and Quality Assurance Standards - Part 3: Guidelines for the Application of ISO 9001 to the Development, Supply and Maintenance of Software

7. ISO 9000-4 Quality Management and Quality Assurance Standards - Part 4: Guide to Dependability Program Management

8. ISO 9001: Quality Systems - Model for Quality Assurance in Design/Development, Product, Installation and Servicing

9. ISO (DIS) 9001 Quality Systems - Model for Quality Assurance in Design, Development,

Production, Installation and Servicing

10. ISO 9002: Quality Systems - Model for Quality Assurance in Production and Installation

11. ISO (DIS) 9002 Quality Systems - Model for Quality Assurance in Production, Installation and Servicing

12. ISO 9003: Quality Systems - Model for Quality Assurance in Final Inspection and Test

13. ISO (DIS) 9003 Quality Systems - Model for Quality Assurance in Final Inspection and Test

14. ISO 9004: Quality Management and Quality System Elements - Guidelines

15. ISO (DIS) 9004-1 Quality Management and Quality System Elements - Part 1: Guidelines

16. ISO 9004-2: Quality Management and Quality System Elements - Part 2: Guidelines for Services

17. ISO (DIS) 9004-4 Quality Management and Quality System Elements - Part 4: Guidelines for Quality Improvement

18. ISO 10011-1: Guidelines for Auditing Quality Systems - Part 1: Auditing

19. ISO 10011-2: Guidelines for Auditing Quality Systems - Part 2: Qualification Criteria for Quality systems Auditors

20. ISO 10011-3: Guidelines for Auditing Quality Systems - Part 3: Management of Audit Programs

21. ISO 10012-1: Quality Assurance Requirements for Measuring Equipment - Part 1: Metrological Confirmation System for Measuring Equipment

22. ISO (DIS) 10013 Guidelines for Developing Quality Manuals

23. ANSI/IEEE 730 Standard for Software Quality Assurance Plans.

24. ANSI/IEEE 1061 Standard for a Software Quality Metrics Methodology.

25. ANSI/IEEE 1298 Software Quality Management System, Part 1: Requirements.

B.4 Configuration Management

1. ANSI/IEEE 1042 Guide to Software Configuration Management.

2. ANSI/IEEE 828 Standard for Software Configuration Plans.

3. ANSI/IEEE 1219 Standard for Software Maintenance.

B.5 Test and Evaluation

1. ANSI/IEEE 829 Standard for Software Test Documentation.

2. ANSI/IEEE 1008 Standard for Software Unit Testing.

3. ANSI/IEEE 1044 Standard for Classification of Software Errors, Faults, and Failures.

4. ANSI/IEEE 1059 Guide for Software Verification and Validation.

B.6 Automated Tools

1. ANSI/IEEE 990 Recommended Practice for ADA as a Program Design Language.

2 ANSI/IEEE 1175 Standard Reference Model for Computing System Tool Interconnections.

3. ANSI/IEEE 1209 Recommended Practices for the Evaluation and Selection of CASE Tools.

4. IEEE P1348 draft 6.0 (1995) Recommended Practices for the Adoption of CASE Tools.

B.7 Human Factors Engineering

1. ANSI/AAMI HE48 Human Factors Engineering, Guidelines and Preferred Practices for the Design of Medical Devices

2. (draft) Laboratory Instruments and Data Management Systems: Design of Software User Interfaces and Software Systems Validation, Operation, and Monitoring; Proposed Guideline NCCLS GP19-P, vol. 14, no. 14, 1994. [available from NCCLS, 771 East Lancaster Avenue, Villanova, PA 19085, 610.525.2435 (voice), 610.527.8399 (fax)]

3. ANSI Z535.3-1991 Criteria for Safety Symbols.

APPENDIX C. Bibliography

This appendix cites books used in the preparation of this document and suggested additional readings on the topics covered. While this list includes many of the current publications, it is not an exhaustive list.

C.1 General Life Cycle Activities

Blum, Bruce I. TEDIUM and the Software Process, MIT Press, Cambridge, Mass. 1990.

Blum, Bruce I. Software Engineering: A Holistic View, Oxford University Press, New York, 1992.

Budgen, David Software Design, Addison-Wesley, 1994, ISBN 0-201-54403-2.

Calvez, Jean Paul Embedded Real-Time Systems: A Specification and Design Methodology. John Wiley & Sons, 1993. ISBN 0-471-93563-8.

Edwards, Keith Real-Time Structured Methods: Systems Analysis John Wiley & Sons, 1993. ISBN-0471-93415-1.

Fairley, Richard Software Engineering Concepts, McGraw-Hill, Inc., 1985, ISBN 0-07-019902-7.

Jackson, Michael Software Requirements and Specifications, Addison-Wesley, 1995, ISBN 0-201-87712-0.

McConnell, Steve Rapid Develoment, Taming Wild Software Schedules, Microsoft Press, Redmond, Washington, 1996

Pressman, R.S. Software Engineering: A Practitioner's Approach, McGraw-Hill, Inc., 1992.

Shumate and Keller Software Specification and Design: A Disciplined Approach for Real-Time Systems. John Wiley & Sons, 1992. ISBN 0-471-53296-7.

Sommerville, Ian Software Engineering, 5th edition, Addison-Wesley, 1996, ISBN 0-201-42765-6.

van Vliet, Hans Software Engineering: Principles and Practices, John Wiley & Sons, Ltd., 1993, ISBN 0-471-93611-1.

Witt, Bernard, Baker, F. Terry, and Merritt, Everett W. Architecture and Design, Van Nostrand Reinhold, 1994, ISBN 0-442-01556-9.

C.2 Safety and Reliability

Leveson, Nancy G. Safeware, Addison-Wesley, 1995, ISBN 0-201-11972-2.

Musa, Iannino & Okumoto Software Reliability Measurement, Prediction, and Application, McGraw-Hill, Inc., 1987.

Musa, J.D. Operational Profiles in Software Reliability Engineering, IEEE Software, vol. 10, no. 2, March 1993, pp. 14-32.

Neumann, Peter G. Computer-Related Risks, ACM Press/Addison Wesley, 1994. ISBN 0-201-55805-x.

Peterson, James L. Petri-Net Theory and the Modeling of Systems, Prentice-Hall, 1981.

Raheja, Dev G. Assurance Technologies, Principles and Practices, McGraw-Hill, Inc., 1991, ISBN 0-07-051212-4.

Roland & Moriarity System Safety Engineering and Management, 2nd edition, Wiley Interscience, 1990. ISBN 0-471-61816-0.

C.3 Quality Assurance

Cho, Chin-Kuei An Introduction to Software Quality Control John Wiley & Sons, Inc., 1980, ISBN 0-471-04704-X.

Schmauch, C. H. ISO 9000 for Software Developers, ASQC Press 1994, ISBN 0-87389-246-1.

Schulmeyer, Gordon, McManus. Handbook of Software Quality Assurance, 2nd edition, Van Nostrand Reinhold, 1992, ISBN 0-442-00796-5.

Schulmeyer, Gordon, McManus. Total Quality Management for Software, Van Nostrand Reinhold, 1993, ISBN 0-442-00794-9.

C.4 Test and Evaluation

Gilb & Graham. Software Inspection, Addison-Wesley, 1993, ISBN 0-201-63181-4.

Habayeb, Abdul System Effectiveness, Naval Post-Graduate School.

Myers, Glenford J. The Art of Software Testing, John Wiley & Sons, Inc., 1979, ISBN 0-471-04328-1.

C.5 Human Factors Engineering

Bias and Mayhew Cost Justifying Usability, Academic Press, 1994.

Brown, Martin L. Human Computer Interface Design Guidelines, Ablex Publishing Co., 1989.

Karat, C. "Cost Justifying Support on Software Development Projects", Human Factors Society Bulletin, Human Factors Society, 1992.

Norman, Donald A. The Psychology of Everyday Things, Basic Books, 1988.

C.6 FDA Publications

These publications may be obtained from the FDA/CDRH/OHIP Division of Small Manufacturers Assistance (DSMA) at 1.800.899.0381 or 1.301.443.7491.

CBER Guideline for the Validation of Blood Establishment Computer Systems, version 1.0, October 1994.

CBER Docket No. 91N-0450, Guideline for Quality Assurance in Blood Establishments.

CDRH Blue Book Memo "Device Labeling Guidance #G91-1", dated March 8, 1991.

CDRH Deciding When to Submit a 510(k) for a Change to an Existing Device.

CDRH FDA Policy for the Regulation of Computer Products, draft, 13 November 1989

CDRH FDA's (draft) Guidance for Off-the-Shelf Software Use in Medical Devices

CDRH General Principles of Software Validation.

CDRH Guidance for the Content and Review of 510(k) Notifications for Picture Archiving and Communications Systems (PACS) and Related Devices, draft August 1993.

CDRH Quality System regulation.

CDRH/OHIP Do It By Design: An Introduction to Human Factors in Medical Devices, draft, February 1996.

ORA/DFI Guide to the Inspections of Software Development Activities (The Software Life cycle), draft, November 1995. (Note: This document was never released by ORA and cannot be obtained.)

ORA Guideline on the General Principles of Process Validation.

ORA FDA Glossary of Computerized System and Software Development Terminology.

Revision Log

Version	Dated	Section(s)	Pages	Comments
Version 1	5/14/98	all	all	Final Release