List of Titles

American National Standard

ANSI/AAMI SW68:2001

Medical device software—Software life cycle processes

AAMI Association for the Advancement of Medical Instrumentation

The Objectives and Uses of AAMI Standards and Recommended Practices

It is most important that the objectives and potential uses of an AAMI product standard or recommended practice are clearly understood. The objectives of AAMI's technical development program derive from AAMI's overall mission: the advancement of medical instrumentation. Essential to such advancement are (1) a continued increase in the safe and effective application of current technologies to patient care, and (2) the encouragement of new technologies. It is AAMI's view that standards and recommended practices can contribute significantly to the advancement of medical instrumentation, provided that they are drafted with attention to these objectives and provided that arbitrary and restrictive uses are avoided.

A voluntary *standard* for a *medical device* recommends to the manufacturer the information that should be provided with or on the product, basic safety and performance criteria that should be considered in qualifying the device for clinical use, and the measurement techniques that can be used to determine whether the device conforms with the safety and performance criteria and/or to compare the performance characteristics of different products. Some standards emphasize the information that should be provided with the device, including performance characteristics, instructions for use, warnings and precautions, and other data considered important in ensuring the safe and effective use of the device in the clinical environment. Recommending the disclosure of performance characteristics often necessitates the development of specialized test methods to facilitate uniformity in reporting; reaching consensus on these tests can represent a considerable part of committee work. When a drafting committee determines that clinical concerns warrant the establishment of *minimum* safety and performance criteria, referee tests must be provided and the reasons for establishing the criteria must be documented in the rationale.

A *recommended practice* provides guidelines for the use, care, and/or processing of a medical device or system. A recommended practice does not address device performance *per se*, but rather procedures and practices that will help ensure that a device is used safely and effectively and that its performance will be maintained.

Although a device standard is primarily directed to the manufacturer, it may also be of value to the potential purchaser or user of the device as a fume of reference for device evaluation. Similarly, even though a recommended practice is usually oriented towards health care professionals, it may be useful to the manufacturer in better understanding the environment in which a medical device will be used. Also, some recommended practices, while not addressing device performance criteria, provide guidelines to industrial personnel on such subjects as sterilization processing, methods of collecting data to establish safety and efficacy, human engineering, and other processing or evaluation techniques; such guidelines may be useful to health care professionals in understanding industrial practices.

In determining whether an AAMI standard or recommended practice is relevant to the specific needs of a potential user of the document, several important concepts must be recognized:

All AAMI standards and recommended practices are *voluntary* (unless, of course, they are adopted by government regulatory or procurement authorities). The application of a standard or recommended practice is solely within the discretion and professional judgment of the user of the document.

Each AAMI standard or recommended practice reflects the collective expertise of a committee of health care professionals and industrial representatives, whose work has been reviewed nationally (and sometimes internationally). As such, the consensus recommendations embodied in a standard or recommended practice are intended to respond to clinical needs and, ultimately, to help ensure patient safety. A standard or recommended practice is limited, however, in the sense that it responds generally to perceived risks and conditions that may not always be relevant to specific situations. A standard or recommended practice is an important *reference* in responsible decision-making, but it should never *replace* responsible decisionmaking.

Despite periodic review and revision (at least once every five years), a standard or recommended practice is necessarily a static document applied to a dynamic technology. Therefore, a standards user must carefully review the reasons why the document was initially developed and the specific rationale for each of its provisions. This review will reveal whether the document remains relevant to the specific needs of the user.

Particular care should be taken in applying a product standard to existing devices and equipment, and in applying a recommended practice to current procedures and practices. While observed or potential risks with existing equipment typically form the basis for the safety and performance criteria defined in a standard, professional judgment must be used in applying these criteria to existing equipment. No single source of information will serve to identify a particular product as "unsafe". A voluntary standard can be used as one resource, but the ultimate decision as to product safety and efficacy must take into account the specifics of its utilization and, of course, cost-benefit considerations. Similarly, a recommended practice should be analyzed in the context of the specific needs and resources of the individual institution or firm. Again, the rationale accompanying each AAMI standard and recommended practice is an excellent guide to the reasoning and data underlying its provision.

In summary, a standard or recommended practice is truly useful only when it is used in conjunction with other sources of information and policy guidance and in the context of professional experience and judgment.

INTERPRETATIONS OF AAMI STANDARDS AND RECOMMENDED PRACTICES

Requests for interpretations of AAMI standards and recommended practices must be made in writing, to the Manager for Technical Development. An official interpretation must be approved by letter ballot of the originating committee and subsequently reviewed and approved by the AAMI Standards Board. The interpretation will become official and representation of the Association only upon exhaustion of any appeals and upon publication of notice of interpretation in the "Standards Monitor" section of the *AAMI News*. The Association for the Advancement of Medical Instrumentation disclaims responsibility for any characterization or explanation of a standard or recommended practice which has not been developed and communicated in accordance with this procedure and which is not published, by appropriate notice, as an *official interpretation* in the *AAMI News*.

American National Standard ANSI/AAMI SW68:2001

Medical device software—
Software life cycle processes

Developed by
Association for the Advancement of Medical Instrumentation

Approved 5 June 2001 by
American National Standards Institute, Inc.

Abstract: Defines the life cycle requirements for medical device software. The set of processes, activities, and tasks described in this standard establishes a common framework for medical device software life cycle processes, with well-defined terminology that can be referenced by the medical device software industry.

Keywords: software, medical device, software life cycle process, software engineering

AAMI Standard

This Association for the Advancement of Medical Instrumentation (AAMI) standard implies a consensus of those substantially concerned with its scope and provisions. The existence of an AAMI standard does not in any respect preclude anyone, whether they have approved the standard or not, from manufacturing, marketing, purchasing, or using products, processes, or procedures not conforming to the standard. AAMI standards are subject to periodic review, and users are cautioned to obtain the latest editions.

CAUTION NOTICE: This AAMI standard may be revised or withdrawn at any time. AAMI procedures require that action be taken to reaffirm, revise, or withdraw this standard no later than 5 years from the date of publication. Interested parties may obtain current information on all AAMI standards by calling or writing AAMI.

All AAMI standards, recommended practices, technical information reports, and other types of technical documents developed by AAMI are *voluntary*, and their application is solely within the discretion and professional judgment of the user of the document. Occasionally, voluntary technical documents are adopted by government regulatory agencies or procurement authorities, in which case the adopting agency is responsible for enforcement of its rules and regulations.

Published by

Association for the Advancement of Medical Instrumentation
1110 N. Glebe Road, Suite 220
Arlington, VA 22201-4795

Printed in the United States of America

ISBN 1–57020–161–7

List of Titles

Contents

Annexes

Figures

Committee representation

Association for the Advancement of Medical Instrumentation

Medical Device Software Committee

This standard was developed by the AAMI Medical Device Software Committee. Committee approval of the standard does not necessarily imply that all committee members voted for its approval.

At the time this document was published, the **AAMI Medical Device Software Committee** had the following members:

Cochairs:	Sherman Eagles
	John F. Murray, Jr.
Secretary:	Nancy George
Members:	Mike Blomquist, Sims Deltec Inc.
	Robert G. Britain, National Electrical Manufacturers Association (NEMA)
	Warren P. Dickinson, Griffith Micro Science Inc.
	Sherman Eagles, Medtronic Inc.
	Brian J. Fitzgerald, Underwriters Laboratories Inc.
	Christine M. Flahive, Chris Flahive Associates, Belle Mead, NJ
	Richard C. Fries, Datex-Ohmeda Inc.
	Larry A. Fry, Siemens Medical Systems
	Nancy George, Software Quality Management Inc., Towson, MD
	Lori Haller, Steris Corp.
	Neil Holland, Abbott Laboratories
	Christopher Keegan, Welch Allyn Inc.
	Alan Kusinitz, Software CPR, Winchester, MA
	Carla Langdon-Sivak, Bard Electrophysiology
	Bernard Liebler, Advanced Medical Technology Association (AdvaMed)
	Don Lin, PhD, Cardiac Science Inc.
	Patrick J. McCormick, PhD, Bausch & Lomb Inc.
	Mary Beth McDonald, St. Jude Medical Inc.
	Martin Morgan, Allegiance Healthcare Corp.
	E. Paul Morozoff, Spacelabs Medical Inc.
	John F. Murray, Jr., U.S. Food and Drug Administration
	Robert M. Sacks, Olympus America Inc.
	Christine Strysik, Baxter Healthcare Corp.
	Joe Veasey, Hill-Rom Air Shields
	Steven D. Walter, Becton Dickinson
Alternates:	Frederick J. Geheb, PhD, Siemens Medical Systems
	Larry Gillum, Allegiance Healthcare Corp.
	Tony Kimpo, Medtronic Physio-Control Corp.
	Gretel Lumley, Agilent Technologies
	Dennis Mertz, Becton Dickinson
	Carl A. Pantiskas, Spacelabs Medical Inc.
	Harvey Rudolph, Underwriters Laboratories, Inc.
	Robert Smith, St. Jude Medical Inc.
	Fayez Sweiss, Abbott Laboratories
	Donna-Bea Tillman, PhD, U.S. Food and Drug Administration
	Stephen Vastagh, National Electrical Manufacturers Association (NEMA)
	Frank Zelina, Steris Corp.

At the time this document was published, the **Executive Board of the AAMI Medical Device Software Committee** had the following members: Sherman Eagles, Brian Fitzgerald, Nancy George, Alan Kusinitz, Bernard Liebler, and John F. Murray.

NOTE—Participation by federal agency representatives in the development of this standard does not constitute endorsement by the federal government or any of its agencies.

Foreword

The International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) published ISO/IEC 12207, *Information technology—Software life cycle processes* in August 1995. The ISO and IEC collaboration that produced this standard is known as Joint Technical Committee 1—Information Technology, Subcommittee 7, Software Engineering.

ISO/IEC 12207 was written as a software life cycle processes standard that could be used in any business sector (aerospace, automotive, medical, etc.). The standard as written contains a comprehensive set of software life cycle processes, activities, and tasks.

The Medical Device Software Committee of the Association for the Advancement of Medical Instrumentation (AAMI) has drafted a standard for use in the medical device business sector based on the framework established in ISO/IEC 12207.

This medical device software standard uses the framework for software life cycle processes established by ISO/IEC 12207 to describe the set of required processes, activities, and tasks necessary to convey this knowledge of intention and provide this demonstration of implementation. In general, this set is less comprehensive than the set defined in ISO/IEC 12207. However, where appropriate, the requirements of ISO/IEC 12207 have been added to and modified.

For the purposes of this standard:

— “shall” means that compliance with a requirement is mandatory for compliance with this standard;

— “should” means that compliance with a requirement is recommended but is not mandatory for compliance with this standard;

— “may” is used to describe a permissible way to achieve compliance with a requirement;

— “establish” means to define, document, and implement; and

— “as appropriate” is defined to be “appropriate” unless the manufacturer can document justification otherwise.

Introduction

Software is often an integral part of medical device technology. Establishing the safety and effectiveness of a medical device containing software requires knowledge of what the software is intended to do and demonstration that the use of the software fulfills those intentions without causing any unacceptable risks.

A minimum set of processes, activities, and tasks must be performed to enable a manufacturer to conduct a medical device risk analysis to determine whether the software can cause a hazard or mitigate a hazard. If the medical device risk analysis determines that the software can cause or mitigate a hazard, additional processes, activities, and tasks are required. See 1.3 of this medical device software standard for additional details.

List of Titles

Medical device software—Software life cycle processes

1 Scope

1.1 Purpose

This standard defines the life cycle requirements for medical device software. The set of processes, activities, and tasks described in this standard establishes a common framework for medical device software life cycle processes, with well-defined terminology that the medical device software industry can reference.

1.2 Field of application

This standard applies to the development and maintenance of medical device software.

This standard may be used when software is a stand-alone medical device or when software is an embedded or integral part of the final medical device.

Medical device development requires the execution of a number of system and subsystem life cycle processes. When a medical device contains software, the software comprises one or more of the subsystems of the medical device.

This standard provides requirements for the life cycle processes, activities, and tasks required for software subsystems of medical devices. It covers the life cycle of medical device software from assignment of system requirements to the software subsystem until the software is no longer in use.

This standard groups software engineering activity into two primary life cycle processes (development and maintenance) and five supporting life cycle processes. Each life cycle process is further divided into a set of activities, with each activity further divided into a set of tasks.

A supporting life cycle process is designed to support another process as an integral part with a distinct purpose and contributes to the success and quality of the software project. A supporting process is used as needed during the execution of another process. For medical device software, the five supporting processes are:

a) software hazard management;

b) documentation;

c) configuration management;

d) verification; and

e) problem resolution.

Activities and tasks required to conduct each of these primary and supporting processes are defined in this standard.

Figure 1 shows the relationship of the primary and supporting processes.

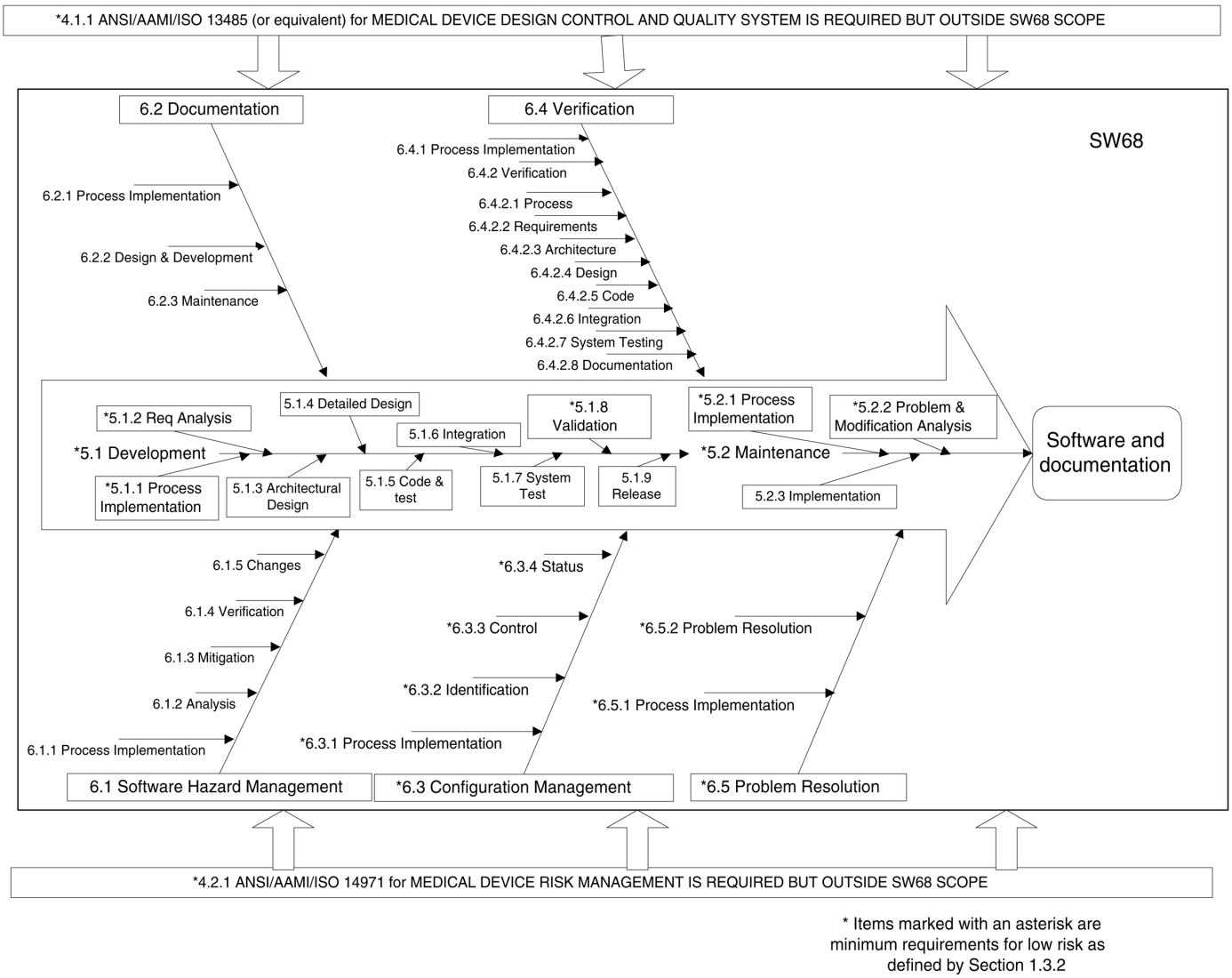

Figure 1—Process and activity summary

1.3 Compliance

Whether software can cause a hazard is determined during the hazard identification activity of the risk management process. The decision to use software to control risk is made during the risk control activity of the risk management process. See ANSI/AAMI/ISO 14971 (normative reference 2.2).

Compliance with this standard is defined as performing all of the processes, activities, and tasks identified in this standard in accordance with 1.3.1 or 1.3.2. Hazards that may be indirectly caused by software (for example, by providing misleading information that could cause inappropriate treatment to be administered) need to be considered when determining whether to select compliance with 1.3.1 or with 1.3.2.

Although the specified processes, activities, and tasks shall be performed, flexibility exists in the methods of implementing these processes and performing these activities and tasks.

1.3.1 Compliance for software that can cause a hazard or that controls risk

For medical device software that can cause a hazard prior to risk control measures or is used to control risks, these processes and activities shall be performed:

a) Processes and activities for a quality management system (4.1) and risk management (4.2).

b) All of the activities of primary life cycle processes (clause 5).

c) All of the activities of supporting processes (clause 6).

1.3.2 Compliance for software that cannot cause a hazard and that does not control risk

A rationale for selecting compliance with this section shall be documented. This rationale shall be based on the results of the device risk management as defined in ANSI/AAMI/ISO 14971 (2.2).

Claiming compliance with this section is justified only if no hazards can be caused by software before risk control measures are implemented and if no risk control measures are required in software.

Note that all processes and activities defined in this standard are considered valuable to ensuring the development and maintenance of high quality software. Omission of many of these processes and activities as requirements for software that cannot cause a hazard should not imply that these processes and activities would not be of value or are not recommended; it merely removes the requirement to maintain documented records of their performance. Also note that, in some regulatory environments, the use of this section may be prohibited.

For medical device software that cannot cause a hazard prior to risk control measures and is not used to control risk, these processes and activities shall be performed:

a) Processes and activities for a quality management system (4.1) and risk management (4.2).

b) The development process implementation (5.1.1), software requirements analysis (5.1.2), and software validation (5.1.8) activities of the development process, including those verification process activities that are specifically identified as requirements in 5.1.1 and 5.1.2.

c) The maintenance process implementation (5.2.1) and problem and modification analysis (5.2.2) activities of the maintenance process.

d) All of the activities of the configuration management process (6.3) and the problem resolution process (6.5).

NOTE 1—Medical device verification and validation shall be performed for all devices.

NOTE 2—Documentation shall be produced and controlled as required by the quality system. Documentation is necessary to establish evidence that the required processes, activities, and tasks were performed.

1.4 Limitations

This standard does not specify the details of how to implement or perform the activities and tasks included in the processes, nor does it specify how to decide which different techniques and methods to use.

This standard contains various processes that are applied throughout the life cycle of the software by various organizations, depending on their needs and goals. This standard does not specify which organization is to perform which process, activity, or task. This standard requires only that the process, activity, or task be completed to establish compliance with this standard.

This standard does not prescribe the name, format, or explicit content of the documentation to be produced. This standard requires documentation of tasks, but the decision of how to package this documentation is left to the user of the standard.

This standard does not prescribe a specific life cycle model. The users of this standard are responsible for selecting a life cycle model for the software project and for mapping the processes, activities, and tasks in this standard onto that model.

2 Normative references

The following standards contain provisions that, through reference in this text, constitute provisions of this standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this standard are encouraged to use the most recent editions of the documents indicated below.

2.1 ANSI/AAMI/ISO 13485:1996, *Quality systems—Medical devices—Particular requirements for the application of ISO 9001.*

2.2 ANSI/AAMI/ISO 14971:2000, *Medical devices—Risk management—Application of risk management to medical devices.*

3 Definitions

For the purposes of this American National Standard, the following definitions apply:

3.1 architecture: Organizational structure of a system or component. (IEEE 610.12:1990)

3.2 baseline: Formally approved version of a configuration item, regardless of media, formally designated and fixed at a specific time during the configuration item's life cycle. (ISO/IEC 12207)

3.3 configuration item: Entity within a configuration that satisfies an end use function and that can be uniquely identified at a given reference point. (ISO/IEC 12207)

3.4 developer: Organization that performs development activities (including requirements analysis, design, testing through acceptance) during the software life cycle process. (ISO/IEC 12207)

3.5 evaluation: Systematic determination of the extent to which an entity meets its specified criteria. (ISO/IEC 12207)

3.6 firmware: Combination of a hardware device and computer instructions or computer data that reside as read-only software on the hardware device. The software cannot be readily modified under program control. (ISO/IEC 12207)

3.7 harm: Physical injury, damage, or both to the health of people or damage to property or to the environment. (ISO/IEC Guide 51)

3.8 hazard: Potential source of harm. (ISO/IEC Guide 51)

3.9 life cycle model: Framework containing the processes, activities, and tasks involved in the development, operation, and maintenance of a software product, spanning the life of the system from the definition of its requirements to the termination of its use. (ISO/IEC 12207)

3.10 maintainer: Organization that performs maintenance activities. (ISO/IEC 12207)

3.11 manufacturer: Natural or legal person with responsibility for designing, manufacturing, packaging, or labeling a medical device; assembling a system; or adapting a medical device before it is placed on the market and/or put into service, regardless of whether these operations are carried out by that person or by a third party on that person's behalf. (ANSI/AAMI/ISO 14971)

3.12 medical device: Any instrument, apparatus, appliance, material, or other article, whether used alone or in combination, including the software necessary for its proper application, that is intended by the manufacturer to be used for human beings for:

— diagnosis, prevention, monitoring, treatment, or alleviation of disease;

— diagnosis, monitoring, treatment, alleviation of, or compensation for an injury or handicap;

— investigation, replacement, or modification of the anatomy or of a physiological process; and

— control of conception;

and that does not achieve its principal intended action in or on the human body by pharmacological, immunological, or metabolic means but may be assisted in its function by these means. (ANSI/AAMI/ISO 13485)

3.13 medical device software: Software necessary for the proper operation of a medical device.

3.14 mitigation: Reduction in the severity of a hazard, the likelihood of the occurrence of a hazard, or both.

3.15 OTS software: Off-the-shelf software. A software item that is already developed and generally available and that has not been developed for the purpose of being incorporated into a medical device. OTS software may be usable either "as is" or with modifications.

3.16 process: A set of interrelated activities that transform inputs into outputs.

NOTE—The term "activities" covers use of resources. (ISO 9000)

3.17 quality assurance: All of the planned and systematic activities that are implemented within the quality system and demonstrated as needed to provide adequate confidence that an entity will fulfill requirements for quality.

NOTE 1—There are both internal and external purposes for quality assurance:

a) Internal quality assurance: performed within an organization to provide confidence to management.

b) External quality assurance: performed in contractual situations to provide confidence to the customer or others.

NOTE 2—Some quality control and quality assurance actions are interrelated.

NOTE 3—Unless requirements for quality fully reflect the needs of the user, quality assurance may not provide adequate confidence. (ISO 9000)

3.18 release: Particular version of a configuration item that is made available for a specific purpose (e.g., test release). (ISO/IEC 12207)

3.19 risk: Combination of the probability of occurrence of harm and the severity of that harm. (ISO/IEC Guide 51)

3.20 security: Protection of information and data so that unauthorized people or systems cannot read or modify them and so that authorized persons or systems are not denied access to them. (ISO/IEC 12207)

3.21 software product: Set of computer programs, procedures, and possibly associated documentation and data. (ISO/IEC 12207)

3.22 software item: Any identifiable part of a software product. (ISO 9000-3)

3.23 software system: Integrated collection of software items organized to accomplish a specific function or set of functions.

3.24 software unit: Software item that is not subdivided into other items for the purpose of configuration management or testing.

3.25 software validation: Confirmation by examination and objective evidence that software specifications conform to user needs and to the intended uses of the medical device, and that the particular medical device requirements implemented through software can be consistently fulfilled.

3.26 system: Integrated composite consisting of one or more of the processes, hardware, software, facilities, and people that provides a capability to satisfy a stated need or objective. (ISO/IEC 12207)

3.27 test coverage: Extent to which a test case tests the requirements for the system or software product. (ISO/IEC 12207)

3.28 testability: Extent to which an objective and feasible test can be designed to determine whether a requirement is met. (ISO/IEC 12207)

3.29 traceability: Degree to which a relationship can be established between two or more products of the development process. (IEEE 610.12)

3.30 user: Individual or organization that uses the operational system to perform a specific function. (ISO/IEC 12207)

3.31 verification: Confirmation by examination and provision of objective evidence that specified requirements have been fulfilled.

NOTE 1—In design and development, verification concerns the process of examining the result of a given activity to determine conformity with the stated requirement for that activity.

NOTE 2—"Verified" is used to designate the corresponding status. (ISO 9000)

3.32 version: Identified instance of an item.

NOTE—Modification to a version of a software product that results in a new version and requires configuration management action. (ISO/IEC 12207)

4 General requirements

4.1 Quality management system

This standard addresses only software engineering activities. However, the development of a safe medical device requires the establishment of a quality management system.

Compliance with ANSI/AAMI/ISO 13485, *Quality systems—Medical devices—Particular requirements for the application of ISO 9001*, or its equivalent national regulation or national standard, is required for use of this standard (see A.4.1). Guidance for applying quality management system requirements to software can be found in ISO 9000-3.

NOTE—In the United States, 21 CFR Part 820, *Quality system regulation* is considered equivalent to ANSI/AAMI/ISO 13485 and meets the intent of this requirement.

4.2 Risk management

Risk management is an essential system process during development and maintenance of a medical device.

4.2.1 Device risk management

Compliance with ANSI/AAMI/ISO 14971, *Medical devices—Risk management—Application of risk management to medical devices* is required for use of this standard (see A.4.2).

4.2.2 Software team participation

At least one individual directly involved in the software development shall participate in the device risk management activity to ensure that risks associated with software are adequately addressed.

5 Primary life cycle processes

This clause defines the following primary life cycle processes:

a) Development process; and

b) Maintenance process.

5.1 Development process

This process consists of the following activities:

a) Process implementation;

b) Software requirements analysis;

c) Software architectural design;

d) Software detailed design;

e) Software coding and testing;

f) Software integration and testing;

g) Software system testing;

h) Software validation; and

i) Software release.

5.1.1 Process implementation

This activity consists of the following tasks:

5.1.1.1 The developer shall define or select a software life cycle model appropriate to the scope, magnitude, complexity, and required safety of the software to be developed. The activities, tasks, and deliverables of the software development process shall be mapped onto the software life cycle model phases. The software life cycle model and mapping of activities and tasks shall be documented.

NOTE—These activities and tasks may overlap or interact and may be performed iteratively or recursively. It is not the intent to imply that a waterfall model should be used.

5.1.1.2 The developer shall verify adequate implementation of the life cycle model in accordance with 6.4.2.1.

5.1.1.3 The developer shall establish a plan or plans for conducting the activities of the software development process. The plan or plans shall include, as appropriate, specific computer programming languages, standards, methods, tools, actions, and responsibility associated with the development and verification of all requirements including safety and security.

5.1.1.4 The plan or plans shall include the activities and tasks necessary to manage any OTS software.

5.1.2 Software requirements analysis

This activity consists of the following tasks:

5.1.2.1 The developer shall establish software requirements. The requirements shall include, as appropriate to the medical device software:

a) Functional and capability requirements, including performance, physical characteristics, and environmental conditions under which the software is to perform;

b) Interfaces external to the software; and

c) Safety requirements, including risk control measures for hardware failures and potential software defects, as well as specifications related to methods of operation and maintenance, environmental influences, and injury;

If included in the medical device design, the requirements shall include, as appropriate to the medical device software:

d) Software-driven alarms, warnings, and operator messages;

e) Security requirements, including those related to compromise of sensitive information;

f) Human factors engineering requirements (including those related to support for manual operations, human–equipment interactions, constraints on personnel, and areas needing concentrated human attention) that are sensitive to human errors and training;

g) Data definition and database requirements;

h) Installation and acceptance requirements of the delivered medical device software at the operation and maintenance site or sites;

i) User documentation to be developed;

j) User operation and execution requirements;

k) User maintenance requirements; and

l) Any required national or international standards or regulations.

5.1.2.2 The developer shall establish software requirements for each OTS software component in addition to those listed in 5.1.2.1. These additional requirements shall include the following:

a) Title and manufacturer of the OTS software component to be used;

b) Version level, release date, patch number, and upgrade designation to be installed and tested in this medical device;

c) The system hardware and software necessary to support the proper operation of this OTS software component (e.g., processor type and speed, memory type and size, system software type, communication and display software requirements);

d) Functional and performance requirements to support the risk management process;

e) Interfaces to the OTS software component; and

f) Safety-related, safety-critical, and risk control measure functions dependent on this OTS software component.

5.1.2.3 Each software requirement (from both 5.1.2.1 and 5.1.2.2) shall be uniquely identified and traceable to verification and validation testing.

5.1.2.4 Each safety requirement implemented in software shall be uniquely identified and traceable to the risk control measures identified in the risk control activity of ANSI/AAMI/ISO 14971 (see 4.2.1).

5.1.2.5 The developer shall verify the software requirements in accordance with 6.4.2.2.

5.1.3 Software architectural design

This activity consists of the following tasks that the developer shall perform or support:

5.1.3.1 The developer shall transform the requirements for the medical device software into an architecture that describes the software's structure and identifies the software items. The architecture of the medical device software shall be documented.

5.1.3.2 The developer shall develop and document an architecture for the interfaces between the software items and the components external to the software items (both software and hardware) and for the interfaces between the software items.

5.1.3.3 The developer shall verify the architecture of the software and any interfaces in accordance with 6.4.2.3.

5.1.4 Software detailed design

This activity consists of the following tasks:

5.1.4.1 The developer shall define software units. Software units shall be testable.

5.1.4.2 The developer shall refine each software item into lower levels containing software units, where appropriate.

5.1.4.3 The developer shall develop a detailed design for each software unit of the software item.

5.1.4.4 The developer shall develop and document a detailed design for any interfaces between the software unit and external components (hardware or software), as well as any interfaces between software units.

5.1.4.5 The developer shall verify the software detailed design in accordance with 6.4.2.4.

5.1.5 Software coding and testing

For each software item, this activity consists of the following tasks:

5.1.5.1 The developer shall establish the following:

a) Each software unit;

b) A strategy and methods for verifying each software unit; and

c) Procedures for verifying each software unit.

5.1.5.2 The developer shall verify the software code and test procedures in accordance with 6.4.2.5.

5.1.6 Software integration and testing

The software integration and testing activity consists of the following tasks:

5.1.6.1 The developer shall establish an integration plan to integrate the software units and perform integration testing. The plan shall include the approach, responsibilities, and sequence. The plan shall include all OTS software components.

5.1.6.2 The developer shall integrate the software units and test as the aggregates are developed in accordance with the integration plan.

5.1.6.3 If a program is built using incremental integration methods, sufficient regression testing shall be conducted.

5.1.6.4 The integration and test results shall be documented. Integration test documentation shall include test results, anomalies found, the version of software tested, relevant hardware and software test configurations, relevant test tools, date tested, and identification of the tester.

5.1.6.5 Integration tests shall include test cases that expose software behavior not only in response to the normal case, but also in response to exceptional, stress, and worst-case conditions. Given input combinations may be limited to those combinations that are conceivable during use of the device.

5.1.6.6 The developer shall verify the integration in accordance with 6.4.2.6.

5.1.7 Software system testing

Software system testing consists of the following tasks:

5.1.7.1 The developer shall establish a plan to verify that the software system meets its requirements. The plan shall include the approach, responsibilities, and sequence. The plan shall include all OTS software components.

5.1.7.2 The developer shall establish, for each software requirement, a set of tests (inputs, expected outcomes, test criteria) and procedures for conducting software system testing.

5.1.7.3 For medical devices that are required to comply with specific national or international standards, relevant software system testing shall demonstrate compliance to such standards.

5.1.7.4 Errors detected during software system testing shall be logged, classified, reviewed, and resolved (but not necessarily fixed) prior to release of the software.

5.1.7.5 When changes are made during the course of software system testing, sufficient regression testing shall be conducted to assure that unintended side effects have not been introduced.

5.1.7.6 The software system tests, the method by which results are determined to be correct or incorrect, and the test results shall be documented. Documented test results shall identify the software version tested, hardware, test tools, revision level or revision date of the test cases, date tested, and identification of the tester.

5.1.7.7 The developer shall verify the software system testing in accordance with 6.4.2.7.

5.1.8 Software validation

NOTE—It is acceptable to combine integration testing, software system testing, and software validation into a single plan and set of activities.

This activity consists of the following tasks, which may be performed as part of the medical device validation:

5.1.8.1 The developer shall establish a software validation plan. The plan shall specify the types of tests and activities that will be used to validate the software. The plan shall include all OTS software components.

5.1.8.2 Test requirements, test cases, and test specifications shall be prepared.

5.1.8.3 The test requirements, test cases, and test specifications shall reflect the particular requirements for the specific intended use, known types of intended users in their intended environment, and range of intended platforms.

5.1.8.4 The software validation tests, the method by which results are determined to be correct or incorrect, and the test results shall be documented. Documented test results shall identify the software version tested, hardware, test tools, revision level or revision date of the test cases, date tested, and identification of the tester.

5.1.9 Software release

This activity consists of the following tasks:

5.1.9.1 The developer shall ensure that software testing has been completed and the results evaluated before the software is released.

5.1.9.2 The developer shall document the versions of the software and documentation that are being released.

5.1.9.3 The developer shall document the procedure and environment used to create the released software.

5.1.9.4 The developer shall archive and maintain master copies of the software, source code, and documentation for the life of the device.

5.1.9.5 The developer shall handle, protect, store, label, package, and deliver the software in accordance with established procedures.

5.2 Maintenance process

This process consists of the following activities:

a) Process implementation;

b) Problem and modification analysis; and

c) Modification implementation.

5.2.1 Process implementation

This activity consists of the following tasks:

5.2.1.1 The maintainer shall establish plans and procedures for conducting the activities and tasks of the maintenance process.

5.2.1.2 The maintainer shall establish procedures for receiving, recording, and tracking problem reports and modification requests. Whenever problems are encountered, they shall be recorded and entered into the problem resolution process (6.5).

5.2.1.3 The maintainer shall establish procedures to evaluate and implement upgrades, bug fixes, and patches for OTS software.

5.2.1.4 The maintainer shall implement (or establish organizational interface with) the configuration management process (6.3) for managing modifications to the existing system.

5.2.2 Problem and modification analysis

This activity consists of the following tasks:

5.2.2.1 The maintainer shall analyze the problem report or modification request for its effect on the organization, existing supported systems, and interfacing systems for the following:

a) Type (e.g., corrective, improvement, preventive, or adaptive to new environment);

b) Scope (e.g., size of modification, number of device models affected, supported accessories affected, resources involved, time to modify); and

c) Criticality (e.g., effect on performance, safety, or security).

The analysis shall be documented.

5.2.2.2 The maintainer shall verify reported problems.

5.2.2.3 The maintainer shall conduct an analysis and determine which documentation, software units, and versions thereof require modification or deletion, as well as whether any new items will be produced. New, modified, and deleted items shall be documented.

5.2.2.4 Based on the analysis, the maintainer shall consider options for implementing the modification.

5.2.2.5 The selected modification option shall be reviewed and approved.

5.2.2.6 The maintainer shall document the problem or the modification request, the analysis results, the implementation options, and the selected and approved option.

5.2.3 Modification implementation

This activity consists of the following tasks:

5.2.3.1 The maintainer shall use the development process (5.1) or an established maintenance process to implement the modifications. The requirements of this process shall include the following:

a) Criteria for testing and evaluating the new and modified items (software units, components, and configuration items) of the system shall be defined and documented.

b) The rationale for the amount of regression testing necessary to ensure that the unmodified items of the system were not affected shall be documented.

5.2.3.2 The complete and correct implementation of the new and modified requirements shall be ensured. It also shall be ensured that the original, unmodified requirements were not affected and that all hazard mitigations (whether intentionally modified or not) remain effective and are not unintentionally compromised. The test results shall be documented.

6 Supporting processes

This clause defines the following supporting life cycle processes:

a) Software hazard management process;

b) Documentation process;

c) Configuration management process;

d) Verification process; and

e) Problem resolution process.

6.1 Software hazard management process

This process consists of the following activities:

a) Process implementation;

b) Software hazard analysis;

c) Mitigation of hazards;

d) Verification of mitigations; and

e) Hazard management of software changes.

6.1.1 Process implementation

This activity consists of the following task:

The software developer shall establish a plan to conduct the activities and tasks of the software hazard management process. All OTS software shall be included in the software hazard management process.

6.1.2 Software hazard analysis

This activity shall consist of the following tasks:

6.1.2.1 Software functionality whose failure could result in the hazards identified in the medical device risk analysis activity of ANSI/AAMI/ISO 14971 (see 4.2.1) shall be identified. This identification shall include hazards that could be the direct result of software failure or for which software implements a risk control measure.

6.1.2.2 Potential causes for the failure of the software functionality identified above shall be identified. Potential causes to be considered shall include, as appropriate:

a) Software defects in the identified functionality;

b) Failure or unexpected results from OTS software; and

c) Hardware failures or other software defects that could result in unpredictable software operation.

6.1.3 Mitigation of hazards

This activity consists of the following task:

Specific hardware, software, working environment or user instruction risk control measures shall be defined, documented and reviewed for adequacy.

6.1.4 Verification of mitigations

This activity consists of the following tasks:

6.1.4.1 Each of the mitigations shall be verified, and this verification shall be documented.

6.1.4.2 There shall be documented traceability of software hazards as appropriate:

a) From the hazard to the software functionality;

b) From the software functionality to the specific software cause or failure;

c) From the software cause or failure to the mitigation; and

d) From the mitigation to the verification of the mitigation.

6.1.5 Hazard management of software changes

This activity consists of the following tasks:

6.1.5.1 Changes to the medical device (including hardware, system, software, OTS software, or intended use) shall be analyzed to determine whether:

a) Any new software mitigations for hazards are required;

b) Any new hazards that could be caused by software are added; and

c) Any new software causes for existing hazards are introduced.

6.1.5.2 Changes to the software, including changes to OTS software, shall be analyzed to determine whether the software modification could interfere with existing risk control measures.

6.1.5.3 Relevant hazard management activities defined in 6.1.2, 6.1.3, and 6.1.4 that are based on these analyses shall be performed.

6.2 Documentation process

This process consists of the following activities:

a) Process implementation;

b) Design and development; and

c) Maintenance.

6.2.1 Process implementation

This activity consists of the following task:

A plan, identifying the documents to be produced during the life cycle of the software product, shall be established. For each identified document, the following shall be addressed:

a) Title or name;

b) Purpose;

c) Intended audience of document;

d) Documentation standards applicable for this document; and

e) Procedures and responsibilities for development, review, approval, modification, and configuration management.

6.2.2 Design and development

This activity consists of the following tasks:

6.2.2.1 Each identified document shall be designed and developed in accordance with any applicable documentation standards defined in the plan.

6.2.2.2 The prepared documents shall be reviewed against their documentation standards and approved as defined in the plan.

6.2.3 Maintenance

This activity consists of the following tasks:

6.2.3.1 Change control procedures shall be established in accordance with the configuration management process (6.3).

6.2.3.2 The tasks that are required to be performed when documentation is to be modified shall be performed (5.2). For those documents that are under configuration management, modifications shall be managed in accordance with the configuration management process (6.3).

6.3 Configuration management process

This process consists of the following activities:

a) Process implementation;

b) Configuration identification;

c) Configuration control; and

d) Configuration status accounting.

NOTE—When this process is used on other software products or entities such as documentation, the term “software item” is interpreted accordingly.

6.3.1 Process implementation

This activity consists of the following tasks:

6.3.1.1 A configuration management plan shall be established. The plan shall describe the items to be controlled; the configuration management activities, procedures, and schedule for performing these activities; the organization(s) responsible for performing configuration management and activities; and their relationship with other organizations, such as software development or maintenance.

6.3.1.2 The developer shall establish procedures to control the receipt, installation, and acceptance of each OTS software component.

6.3.2 Configuration identification

This activity consists of the following task:

A scheme shall be established for the unique identification of software configuration items and their versions to be controlled for the project. This scheme shall include OTS software components. For each software configuration item and its versions, the documentation that defines the baseline and the version references shall be identified.

6.3.3 Configuration control

This activity consists of the following tasks:

6.3.3.1 The following tasks shall be performed: identification and recording of change requests; analysis and evaluation of the changes; approval or nonapproval of the request; and implementation, verification, and release of the modified software item. Each upgrade, bug fix, or patch for OTS software shall be evaluated, and the evaluation shall be documented.

6.3.3.2 An audit trail shall exist whereby each modification, the reason for the modification, and authorization of the modification can be traced.

6.3.4 Configuration status accounting

This activity consists of the following task:

Records of the history of controlled items including software baseline shall be retrievable.

6.4 Verification process

This process consists of the following activities:

a) Process implementation; and

b) Verification.

6.4.1 Process implementation

This activity consists of the following tasks:

6.4.1.1 A process shall be established to verify the software product.

6.4.1.2 An appropriately trained or experienced organization or team member shall be selected to be responsible for conducting the verification. This organization or team member shall be granted the independence and authority to perform the verification activities.

6.4.1.3 Life cycle activities and deliverables from each stage of the life cycle requiring verification shall be determined and documented in plans or procedures. Verification activities and tasks defined in 6.4.2, including associated methods, techniques, and tools for performing the tasks, shall be selected for the target life cycle activities and software products and documented in plans or procedures.

6.4.1.4 A verification plan shall be established based upon the verification tasks as determined. The plan shall address the life cycle activities and software products subject to verification; the required verification tasks for each life cycle activity and software product; and related resources, responsibilities, and sequence. For devices for which

a fully independent verification process is not specified, verification planning may be integrated with software development or quality assurance planning.

6.4.1.5 The verification plan shall be implemented. Problems and nonconformances detected by the verification effort shall be entered into the problem resolution process (6.5) if required in plans or procedures. All problems and nonconformances shall be resolved.

NOTE—A problem does not have to be corrected to comply with the problem resolution process, provided that the disposition of the problem has been reviewed and the problem has been evaluated for possible relevance to safety as specified in ANSI/AAMI/ISO 14971.

6.4.2 Verification

This activity consists of the following tasks:

a) Process implementation verification;

b) Software requirements verification;

c) Software architecture verification;

d) Software detailed design verification;

e) Software coding and testing verification;

f) Software integration and testing verification;

g) Software system testing verification; and

h) Document verification.

6.4.2.1 Process implementation verification

For process implementation verification, the following requirements shall be addressed as appropriate:

a) At phase completion, the existence of the required deliverables defined in the plan shall be verified; and

b) The date of each phase transition shall be documented.

NOTE—The phases and deliverables of a software life cycle are defined in 5.1.1.

6.4.2.2 Software requirements verification

6.4.2.2.1 For software requirements verification, the following criteria shall be addressed as appropriate:

a) Consistency;

b) Testability;

c) Traceability to device requirements;

d) Feasibility of software design;

e) Adequacy of hazard mitigation requirements;

f) Feasibility of operation and maintenance; and

g) Traceability to the hazard analysis and the system verification testing.

6.4.2.2.2 The results of software requirements verification shall be documented.

6.4.2.3 Software architecture verification

6.4.2.3.1 For software architecture verification, the following requirements shall be addressed as appropriate:

a) Adequacy for hazard mitigation;

b) Internal consistency among the software items; and

c) Feasibility of interfaces among software items and among software items and hardware.

6.4.2.3.2 The results of the software architecture verification shall be documented.

6.4.2.4 Software detailed design verification

6.4.2.4.1 For software detailed design verification, the following requirements shall be addressed as appropriate:

a) Complete coverage of the requirements of the software item;

b) External consistency with architectural design;

c) Internal consistency among software units;

d) Appropriateness of design methods and standards used;

e) Adequacy of hazard mitigation for hazards created or mitigated by software;

f) Feasibility of testing;

g) Feasibility of operation and maintenance;

When present in the design, the following shall also be addressed as appropriate:

h) Implementation of the intended events, inputs, outputs, interfaces, logic flow, allocation of timing and sizing budgets, and error definition, error isolation, and error recovery;

i) Definition of the default state, in which all faults that may result in a hazard are addressed, with events and transitions;

j) Initialization of variables, memory management; and

k) Cold and warm resets, standby, and other state changes that may affect the hazard mitigation.

6.4.2.4.2 The results of the design verification shall be documented.

6.4.2.5 Software coding and testing verification

6.4.2.5.1 For software coding and testing verification, the following requirements shall be addressed as appropriate:

a) The code is correct and complies with requirements;

b) The code implements safety requirements correctly;

c) The code is consistent with the interfaces documented in the detailed design of the software unit;

d) Coding methods and standards used are appropriate;

e) Conformance with established programming procedures or coding standards;

f) Appropriateness of verification strategies used;

g) Correctness of the test procedures;

h) Feasibility of software integration and testing;

i) Maintainability;

When present in the design, the following shall also be addressed as appropriate:

j) Proper event sequence;

k) Correct data and control flow;

l) Appropriate allocation timing and sizing budgets;

m) Fault handling (error definition, isolation, and recovery);

n) Initialization of variables;

o) Self-diagnostics;

p) Memory management and memory overflows; and

q) Boundary conditions.

6.4.2.5.2 The results of the software coding and testing verification shall be documented.

6.4.2.6 Software integration and testing verification

6.4.2.6.1 For software integration verification, the following requirements shall be addressed as appropriate:

a) The software units of each software item have been integrated into the software item;

b) The hardware items, software items, and support for manual operations of the system have been integrated into the system; and

c) The integration tasks have been performed in accordance with an integration plan.

6.4.2.6.2 For software integration testing, the following requirements shall be addressed as appropriate:

a) The software meets its design objectives;

b) Specified timing and other behavior;

c) Specified functioning of internal and external interfaces;

d) Stress testing is sufficient to identify resource problems; and

e) Anomalies have been documented and understood.

6.4.2.6.3 The results of the software integration and testing verification shall be documented.

6.4.2.7 Software system testing verification

6.4.2.7.1 For software system testing, the following requirements shall be addressed as appropriate:

a) Traceability to the software requirements;

b) Dynamic or static test coverage of the code of each software item;

c) Conformance to expected results;

d) Feasibility of medical device system integration and testing, if conducted separately;

e) Effectiveness of modifications made to correct anomalies; and

f) Maintainability.

6.4.2.7.2 The results of the software system testing shall be documented.

6.4.2.8 Documentation verification

For verification of the documentation defined in plans and procedures, the following requirements shall be addressed as appropriate:

a) The documentation is prepared according to the requirements of plans and procedures and is accurate and controlled;

b) Documentation has been prepared in the sequence specified in plans and procedures; and

c) Configuration management of documents follows specified procedures.

6.5 Problem resolution process

This process consists of the following activities:

a) Process implementation; and

b) Problem resolution.

6.5.1 Process implementation

This activity consists of the following task:

A problem resolution process shall be established for handling all known problems (including nonconformance) detected in the software products and activities. Plans or procedures shall specify at each stage of the life cycle the aspects of the problem resolution process that will be formal and documented. The process shall comply with the following requirements:

a) The process shall be closed loop, ensuring that all detected problems are reported and entered into the problem resolution process; action is initiated on them; relevant parties are advised of the existence of the problem as appropriate; causes are identified, analyzed, and eliminated, where possible; resolution and disposition are achieved; status is tracked and reported; and records of the problems are maintained.

b) The process shall include an evaluation of the possible relevance to safety as specified in ANSI/AAMI/ISO 14971.

c) Analysis shall be performed to detect trends in the problems reported.

d) Problem resolutions and dispositions shall be verified to determine whether problems have been resolved, adverse trends have been reversed, and changes have been correctly implemented in the appropriate software products and activities, as well as to determine whether additional problems have been introduced.

NOTE 1—The process should contain a scheme for categorizing and prioritizing the problems. Each problem should be classified by type, scope, and criticality to facilitate trend analysis and problem resolution.

NOTE 2—A problem does not have to be corrected to comply with the problem resolution process, provided that the disposition of the problem has been reviewed and the problem has been evaluated for possible relevance to safety as specified in ANSI/AAMI/ISO 14971.

6.5.2 Problem resolution

This activity consists of the following task:

When problems (including nonconformances) have been detected in a software product or an activity, a problem report shall be prepared to describe each problem detected. The problem report shall be used as part of the closed-loop process described above from detection of the problem through investigation, analysis, and resolution of the problem and its cause, and to trend detection across problems.

Annex A
(informative)

Rationale for the development and guidance on the provisions of this standard

A.1 Scope

A.1.1 Purpose

The U.S. Food and Drug Administration (FDA) may recognize consensus national or international standards for use in regulating medical devices. The FDA has indicated that some software standards may be useful in reducing the material that must be submitted for premarket review of medical devices containing software. The purpose of this standard is to reduce the time required for regulatory review of medical device software by reducing the material that must be reviewed while providing a development process that will consistently produce high quality, safe medical device software. To accomplish this, a medical device software standard is needed that identifies the minimum activities and tasks that need to be accomplished to provide confidence that the software has been developed in a manner that is likely to produce highly reliable and safe software products.

This standard has been developed because of a perceived need for a standard that specifically addresses medical device software engineering. Many medical devices are rapidly becoming software intensive. It is nearly impossible to measure software to determine if it meets its specifications, yet software provides a great deal of flexibility that can be used to work around hardware or other problems found during development. To create a sound software design and implement it requires a disciplined, methodical approach. This approach is what we mean by software engineering.

A.1.2 Field of application

Many standards address various software development activities, and some standards address software safety for some types of medical devices, but this fragmentation means that manufacturers comply with multiple standards to achieve the desired result. This standard uses ISO/IEC 12207, *Information technology—Software life cycle processes* as a starting point to develop a standard for use in medical device software engineering.

ISO/IEC 12207 contains a comprehensive set of requirements covering both project-oriented and organization-oriented activities, including system development activities and software engineering activities. This standard uses the established ISO/IEC 12207 framework and identifies the minimum set of requirements that are necessary for medical device software engineering.

Some portions of ISO/IEC 12207 are not necessary for development of medical device software because medical device manufacturers are required to have established a quality management system. Some sections of ISO/IEC 12207 duplicate quality management system requirements. Repeating these requirements in a software engineering standard, possibly in a slightly different way, is redundant and possibly confusing. Likewise, medical device manufacturers must have a device development protocol that documents system-level development activities. Repeating requirements for system development activities in this software engineering standard is not necessary. The focus of this standard is on software engineering activities. Quality management system and system development requirements have not been included in this standard.

A.1.3 Compliance

A claim of compliance with this standard must include a documented rationale for selecting compliance with 1.3.1 or 1.3.2. This rationale should be based on the results of the device risk management as defined in ANSI/AAMI/ISO 14971. Claiming compliance with 1.3.2 is justified only if no hazards exist that can be caused by software before risk control measures are implemented and if no risk control measures are required in software.

Hazards that may be indirectly caused by software (for example, by providing misleading information that could cause inappropriate treatment to be administered) should be considered when determining whether to select compliance with 1.3.1 or 1.3.2.

To comply with this standard, the manufacturer must document why any items marked “as appropriate” were not addressed. The location of this documentation is not prescribed, but it likely would be with the documentation of the task being performed.

A.1.3.1 Compliance for software that can cause a hazard or that controls risks

All software processes, activities, and tasks identified in this standard are required for software when a risk analysis shows that the software can cause an injury or when software is used to mitigate a hazard that can cause an injury. Software that a risk analysis shows can cause serious injury or death must comply with these requirements, but these requirements may not be sufficient for this software.

Some examples of devices that might contain software that can cause a hazard or control risks include the following:

— Bedside monitors; and

— In vitro diagnostic devices.

A.1.3.2 Compliance for software that cannot cause a hazard and does not control risk

Before the risk management approach that is taken by standards for safety requirements (such as IEC 60601-1-4 and IEC 61508) can be applied, confidence should be established in the software engineering tasks on which the risk analysis is based. If the software requirements that describe what the device is to do are not adequately documented, or if change control and configuration management are not practiced, little confidence can be created in the results of the risk management. The output of an undisciplined software engineering practice is not an adequate basis for a medical device risk analysis. A minimum set of software engineering process requirements must be followed for every medical device software development before the results of a medical device risk analysis can be considered acceptable. This standard identifies that set of requirements for medical device software engineering. These requirements must be met even if the medical device software cannot cause or mitigate a hazard.

Note that all processes and activities defined in this standard are considered valuable in assuring the development and maintenance of high quality software. The omission of many of these processes and activities as requirements for software that cannot cause a hazard should not imply that these processes and activities would not be of value or are not recommended. Their omission is intended to recognize that software that cannot cause a hazard can be easily assured of medical safety and effectiveness primarily through overall validation activities during the design of a medical device (which is outside the scope of this standard) and through some simple software life cycle controls.

Some examples of devices that might contain software that cannot cause a hazard and does not control risk include the following:

— Digital thermometer;

— Breast pump;

— Biofeedback device; and

— Surgical lamp.

A.2 Normative references

This standard must be used in the context of a quality management system and a risk management process. ANSI/AAMI/ISO 13485 and ANSI/AAMI/ISO 14971 provide the details for these activities that are necessary but outside the scope of this standard. ISO 9000-3 provides guidance for applying a quality management system to software development. This guidance is not required by this standard but is highly recommended.

A.3 Definitions

Where possible, terms have been defined using definitions from international standards.

Identifying terms that provide the flexibility necessary for describing components of software products developed by many different methods has been difficult. The IEEE glossary (IEEE 610.12:1990) contains the following comment:

> NOTE—The terms “module,” “component,” and “unit” are often used interchangeably or defined to be sub-elements of one another in different ways depending upon the context. The relationship of these terms is not yet standardized.

We have chosen to use three terms. The top level is the software system. The software system is a subsystem of the medical device (see IEC 60601-1-4). The lowest level that is not further decomposed for the purposes of testing or configuration management is the software unit. All levels of composition, including the top and bottom levels, may be called software items. A software system, then, is composed of one or more software items, and each software item is composed of one or more software units. The responsibility is left to the developer to provide the definition and

granularity of the software items and software units. Leaving these terms vague allows one to apply them to the many different development methods and types of software used in medical devices.

A.4 General requirements

A.4.1 Quality management system

Two reasons can be stated for requiring the compliance to a quality management system for use of this standard. The first is that medical devices containing software are complex and require the discipline in all areas that is needed to implement and adhere to a quality management system. The second is that the starting document, ISO/IEC 12207, includes organizational processes such as management, infrastructure, improvement, and training. These processes must be in place to develop high quality software. However, these areas are covered by a quality management system. The inclusion of these processes in a software engineering standard could lead to confusion and extra effort for those organizations that have implemented a quality management system. The approach that has been taken is, first, to require that the users of this standard have implemented a quality management system and, second, to focus this standard on software engineering requirements. ANSI/AAMI/ISO 13485 is an approved consensus International Standard that is specifically intended for applying ISO 9001 to medical devices. To gain pre-market approval of a medical device in the United States, a manufacturer must comply with the FDA *Quality System Regulation*, 21 CFR Part 820. For the purpose of this software standard, the *Quality System Regulation* and ANSI/AAMI/ISO 13485 are considered equivalent.

A.4.2 Risk management

Great variation can occur in the tasks and the rigor with which the tasks are performed when developing software. If the software can affect safety, then establishing requirements for the tasks and requirements for the effectiveness of those tasks is appropriate, depending on the software's effect on safety. It is not possible to determine the risk resulting from software independent of the device that uses the software. It is also not possible to determine the risk resulting from the software unless what the software does and how it interfaces with other components of the device have been described and unless changes to other components are controlled. This standard requires a minimum number of activities and tasks to allow analysis of the risk resulting from the software. Risk management must be performed for the device to determine whether the software contributes to risk. Rather than trying to define an appropriate risk management process in this software engineering standard, compliance with the consensus International Standard, ANSI/AAMI/ISO 14971, which deals explicitly with risk management for medical devices, is required. To ensure that the analysis of the risk includes the software, a member of the software team should participate in the risk analysis. Specific software hazard management activities resulting from hazards that have software as a contributing cause are identified in a supporting process of this standard.

A.5 Primary life cycle processes

A.5.1 Development process

The development process consists of the activities and tasks of the developer. The process includes the activities for requirements analysis, software architecture, detailed design, coding and testing, integration, software system testing, software validation, and software release related to medical device software. This software development process assumes that the appropriate system or medical device development life cycle and quality management systems processes are in place; therefore, only software activities and tasks are specified here.

NOTE—Software verification is covered by the supporting life cycle process.

A.5.1.1 Process implementation

The process implementation activity requires the developer to define, document, and implement the life cycle model to be used, as well as the plans for conducting the software development. A life cycle model is needed to relate the activities and tasks of development over time. Defining phases with entry and exit criteria is necessary to plan evaluation of deliverables and determine readiness for performing the next activity. Plans for conducting the activities are required to ensure uniform execution of activities and tasks associated with the development, and to assign responsibility for the execution of these activities and tasks.

A.5.1.2 Software requirements analysis

This activity requires the developer to establish and verify the software requirements for the medical device software. Establishing verifiable requirements is essential for determining what is to be built, for determining that the medical device software exhibits acceptable behavior, and for demonstrating that the completed medical device software is ready for use. To demonstrate that the requirements have been implemented as desired, each requirement must be identifiable and must be stated in such a way that objective criteria can be established to prove that it has been implemented correctly. If the device risk management process imposed requirements on the software to control identified risks, these requirements must be identified in the software requirements in such a way as to make it

possible to trace the risk controls to the software requirements. All software requirements should be identified in such a way as to make it possible to demonstrate traceability between the requirement and software system testing. If regulatory approval in some countries requires conformance to specific regulations or international standards, this conformance requirement should be documented in the software requirements. Because the software requirements establish what is to be implemented in the software, an evaluation of the requirements is required before the requirements analysis activity is complete.

An area of frequent confusion is the distinction between user needs, design inputs, software requirements, software functional specifications, and software design specifications. *Design inputs* are the interpretation of user needs into formally documented medical device requirements. *Software requirements* are the formally documented specifications of what the software must do to meet the user needs and the design inputs. Software *functional specifications* are often included with the software requirements and define in detail what the software must do to meet its requirements even though many different alternatives might also meet the requirements. Software *design specifications* define how the software will be designed and decomposed to implement its requirements and functional specifications.

Traditionally, software requirements, functional specifications, and design specifications have been written as a set of one or more documents. It is now feasible to consider this information as data items within a common database. Each item would have one or more attributes that would define its purpose and linkage to other items in the database. This approach allows presentation and printing of different views of the information best suited for each set of intended users (e.g., marketing, developers, testers, auditors) and supports traceability to demonstrate adequate implementation and test coverage of all requirements. Tools to support this approach can be as simple as a hypertext document using HTML hyperlinks or as complex and capable as CASE tools and requirements analysis tools.

A.5.1.3 Software architectural design

This activity requires the developer to define the major structural components of the software, their externally visible properties, and the relationship among them. If the behavior of a component can affect other components, that behavior should be described in the architecture. This description is especially important for behavior that can affect components of the medical device that are outside the software. Architectural decisions are extremely important for implementing safety requirements. Without understanding (and documenting) the behavior of a component that may affect other components, it will be nearly impossible to show that the system is safe. A software architectural design is necessary to ensure the correct implementation of the software requirements. The software architecture is complete when all software requirements can be allocated to the identified software items. Because the correct design and implementation of the software is dependent on the architecture, the architecture must be verified to complete this activity. Verification of the architecture is generally done by technical review.

A.5.1.4 Software detailed design

This activity requires the developer to refine the software items and interfaces defined in the architecture to create software units and their interfaces. Although software units are often thought of as being a single function or module, this view is not always appropriate. We have defined *software unit* to be a software item that is not subdivided into smaller items. Software units can be tested separately. The developer must define the level of detail of the software unit. Detailed design specifies algorithms, data representations, interfaces among different software units, and interfaces between software units and data structures. Detailed design must also be concerned with the packaging of the software product. It is necessary to document the design of each software unit and its interface so that the software unit can be implemented correctly. The detailed design fills in the details necessary to construct the software. It should be complete enough that the programmer is not required to make ad hoc design decisions. Because correct implementation depends on correct detailed design, it is necessary to verify the detailed design before the activity is complete. Verification of detailed design is generally done by technical review.

A.5.1.5 Software coding and testing

This activity requires the developer to write and verify the code for the software units. The detailed design must be translated into source code. Coding represents the point where decomposition of the specifications ends and composition of the executable software begins. To consistently achieve the desirable code characteristics, coding standards should be used to specify a preferred coding style. Examples of coding standards include requirements for understandability, language usage rules or restrictions, and complexity management. The code for each unit must be verified to ensure that it functions as specified by the detailed design and that it complies with the specified coding standards.

A.5.1.6 Software integration and testing

This activity requires the developer to plan and execute integration of software units into aggregate software items and to verify that the software items behave as intended.

The approach to integration may range from nonincremental integration to any form of incremental integration. The properties of the software item being assembled dictate the chosen method of integration.

Software integration testing focuses on the transfer of data and control across a software item's internal and external interfaces. External interfaces are those with other software, including operating system software and medical device hardware.

The rigor of integration testing and the level of detail of the documentation associated with integration testing should be commensurate with the risk associated with the device, the device's dependence on software for potentially hazardous functions, and the role of specific software items in higher risk device functions. For example, although all software items should be tested, items that have an effect on safety should be subject to more direct, thorough, and detailed tests.

As applicable, integration testing demonstrates program behavior at the boundaries of its input and output domains and confirms program responses to invalid, unexpected, and special inputs. The program's actions are revealed when given combinations of inputs or unexpected sequences of inputs, or when defined timing requirements are violated. The test requirements in the plan should include, as appropriate, the types of white box testing to be performed as part of integration testing.

The plans and test documentation identified in 5.1.6 through 5.1.8 may be individual documents tied to specific phases of development or evolutionary prototypes. They also may be combined so a single document or set of documents covers the requirements of multiple subsections. All or portions of the documents could be incorporated into higher level project documents such as a software or project quality assurance plan or a comprehensive test plan that addresses all aspects of testing for hardware and software. In these cases, a cross reference should be created that identifies how the various project documents relate to each of the software integration tasks.

Software integration testing may be performed in a simulated environment, on actual target hardware, or on the full medical device.

A.5.1.7 Software system testing

This activity requires the developer to verify the software's functionality by verifying that the requirements for the software have been successfully implemented.

Software system testing demonstrates that the specified functionality exists. This testing verifies the functionality and performance of the program as built with respect to the requirements for the software.

Software system testing focuses on functional (black box) testing, although it may be desirable to use white box methods to more efficiently accomplish certain tests, initiate stress conditions or faults, or increase code coverage of the qualification tests. The organization of testing by types and test stage is flexible, but coverage of requirements, hazard mitigation, usability, and test types (e.g., fault, installation, stress) should be demonstrated and documented.

Software system testing tests the integrated software and may be performed in a simulated environment, on actual target hardware, or on the full medical device.

When a change is made to a software system (even a small change), the degree of regression testing (not just the testing of the individual change) should be determined to ensure that no unintended side effects have been introduced. This regression testing (and the rationale for not fully repeating software system testing) should be planned and documented.

Software system test responsibilities may be dispersed, occurring at different locations and being conducted by different organizations. However, regardless of the distribution of tasks, contractual relations, source of components, or development environment, the device manufacturer retains ultimate responsibility for ensuring that the software functions properly for its intended use.

If anomalies were uncovered during testing that were verified, but a decision was made not to fix them, then these anomalies must be reviewed in relation to the hazard analysis to verify that they do not affect the safety and efficacy of the device. The root cause, symptoms, and rationale for not fixing them should be documented.

A.5.1.8 Software validation

Validation is a confusing term because it is defined and used in many ways by different industries and in different standards. In some sense, this standard could have been named "Software validation for medical device software." Use of the term in this sense would refer to all formal aspects of software development that together ensure safe and effective software.

Many in the software industry use the term *software validation* to refer to final testing of the software on its intended platform or platforms. The purpose of this testing is to verify that the software is suitable for use in its intended environment and by its intended users. This use of the term is intended in this standard.

Neither these nor other definitions should be considered right or wrong. What is important is to understand all of the activities and process controls necessary to ensure that safe, effective, reliable software is produced.

Depending on the circumstances, controlled software validation testing at (potential) customer locations may be used. In this case, test plans should identify the controls to ensure safety, that the intended coverage is achieved, and that proper documentation is prepared when testing is conducted at sites not directly controlled by the software developer.

Software validation differs from software system testing because software validation verifies the suitability of the software system for use in its intended environment and by its intended users. Software system testing verifies only that the requirements for the software have been successfully implemented. Software validation and software system testing may, of course, be combined into a single plan and set of activities.

Overall validation of medical device design is assumed as part of the manufacturer's quality system but is outside the scope of this standard. However, it may be combined with software system testing or software validation, provided that the testing is performed adequately and covers the aspects of software testing defined in this standard. Combining these activities may be most appropriate when the device itself is a stand-alone software package or an information system.

A.5.1.9 Software release

This activity requires the developer to document the version of the medical device software being released, specify how it was created, and follow appropriate procedures for release of the software. Software release covers the transition of the software product from the development phase to a completed product ready for distribution.

The developer must be able to show that the software that was developed and qualified using the development process is the software that is being released. The developer must also be able to retrieve the master copy of the software in case it is needed in the future and must store, package, and deliver the software in a manner that prevents the software from being damaged or misused. Defined procedures should be established to ensure that these tasks are performed appropriately and with consistent results.

A.5.2 Maintenance process

A.5.2.1 Process implementation

This activity requires the maintainer to plan maintenance activities and tasks and to document procedures for executing them. To implement corrective actions, control changes during maintenance, and manage release of revised software, the maintainer must record and resolve problem reports and requests from users, as well as manage modifications to the medical device software. The maintenance process includes the activities and tasks of the maintainer. This process is activated when the medical device software undergoes modifications to code and associated documentation because of either a problem or the need for improvement or adaptation. The objective is to modify existing medical device software while preserving its integrity. This process includes the migration of the medical device software. The activities provided in this clause are specific to the maintenance process; however, the maintenance process may use other processes in this standard.

A.5.2.2 Problem and modification analysis

This activity requires the maintainer to analyze problem reports and modification requests for their effect; verify reported problems; and consider, select, and obtain approval for implementing a modification option. Problems and modification requests may affect the performance, safety, or regulatory clearance of a medical device. An analysis is necessary to determine whether any effects exist because of a problem report or whether any effects will result from a modification to correct a problem or implement a request. It is especially important to verify through trace or regression analysis that the risk control measures built into the device are not changed or modified by the software change that is being implemented as part of the software maintenance activity. It is also important to verify that the modified software does not cause or mitigate a hazard in software that previously did not cause or mitigate hazards. In the event that the software modification now may cause or mitigate a hazard, the compliance to this standard must be demonstrated as in 1.3.2. Because of the potential effects, management personnel who are responsible for overseeing these areas should approve implementation of a modification option.

A.5.2.3 Modification implementation

This activity requires that the maintainer first determine what must be modified and then use an established process to make the modification. If a maintenance process has not been defined, the appropriate development process tasks may

be used to make the modification. The maintainer must also ensure that the modification does not cause a negative effect on other parts of the medical device software. Unless the medical device software is treated as a new development, analysis of the effect of a modification on the entire medical device software is necessary. A rationale must be made that justifies the amount of regression testing that will be performed to ensure that the portions of the medical device software not being modified still perform as they did before the modification was made.

A.6 Supporting processes

A.6.1 Software hazard management process

Software hazard management is a part of overall medical device risk management and cannot be adequately addressed in isolation. This standard requires the use of ANSI/AAMI/ISO 14971 for risk management. Risk management as defined in ANSI/AAMI/ISO 14971 deals specifically with risk to safety. One portion of ANSI/AAMI/ISO 14971 pertains to control of identified risks associated with each hazard identified during the risk analysis. The software hazard management process in this standard is intended to provide additional requirements for risk control for software, including software that has been identified during the risk analysis as potentially causing a hazard, or software that is used to control medical device risks. The software hazard management process is included in this standard for two reasons:

a) The intended audience of this standard includes software developers, groups, and vendors who need to understand minimum requirements for risk control measures in their area of responsibility—software.

b) The general risk management standard, ANSI/AAMI/ISO 14971, provided as a normative reference in this standard, does not adequately address software-specific risk control activities and their relationship to the software development life cycle.

Software hazard management is a part of overall medical device risk management. Plans, procedures, and documentation required for the software hazard management activities can be a series of separate documents or a single document, or they can be integrated with the medical device risk management activities and documentation as long as all requirements in this standard are met.

As discussed in ANSI/AAMI/ISO 14971, software failures are systemic in nature. The accurate estimation of systemic failure rates is difficult. In cases where an appropriate level of confidence cannot be established for the estimation of systemic failures, the rigor of the software hazard management process, the level of detail of associated documentation, and the adequacy of mitigations should be determined according to the severity of harm that could result from the hazards related to software. Because parts of the software not directly implementing critical safety functions or mitigations can still have side effects on the safety-related software, the most severe hazard that could be caused by software should be used as the determining factor in defining the rigor of the development process and associated hazard mitigations. This determination can be made by defining the clinical details of the worst-case hazards or by simply rating these hazards along some general severity scale such as the following:

— *Major*: The severity is major if operation of the software associated with device function directly affects the patient and/or the operator in such a way that failures or latent flaws could result in death or serious injury to the patient and/or the operator; or if operation of the software indirectly affects the patient and/or the operator (e.g., through the action of the care provider) in such a way that incorrect or delayed information could result in death or serious injury of the patient and/or the operator.

— *Moderate:* The severity is moderate if the operation of the software associated with device function directly affects the patient and/or the operator in such a way that failures or latent design flaws could result in nonserious injury to the patient and/or the operator; or if operation of the software indirectly affects the patient and/or the operator (e.g., through the action of a care provider) in such a way that incorrect or delayed information could result in nonserious injury of the patient and/or the operator.

— *Minor:* The severity is minor if failures or latent design flaws would not result in any injury to the patient and/or the operator.

Serious injury is defined as an injury or illness that:

— is life threatening;

— results in permanent impairment (i.e., irreversible impairment or damage to a body structure or function excluding trivial impairment or damage) of a body function or permanent damage to a body structure; or

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

A.6.2 Documentation process

The documentation process is a process for recording information produced by a software life cycle process or activity. The process includes the set of activities related to planning, designing, developing, producing, editing, distributing, and maintaining those documents required by all concerned, such as managers, engineers, and users of the system or software product.

A.6.2.1 Process implementation

This activity requires the developer to plan the documentation that will be produced during the software development. This standard does not require specific documents but does require documentation produced by various tasks. The developer has the responsibility to identify which documents that are being produced contain the required documentation. Responsibilities, procedures, and documentation standards are needed to ensure that the documentation required by this standard is produced and verified in an acceptable manner.

A.6.2.2 Design and development

This activity requires the developer to produce the planned documentation and have it reviewed and approved. This standard requires that documentation be produced by many tasks. This documentation must be developed and published in accordance with the documentation plan and other appropriate plans.

A.6.2.3 Maintenance

This activity requires the developer to use the maintenance process tasks as necessary to maintain documentation and requires that change control be established for documentation changes. Any documentation that is part of the medical device software may have problem reports or modification requests after release. These reports and requests must be planned and managed in the same way that software modifications are planned and managed.

A.6.3 Configuration management process

The configuration management process is a process of applying administrative and technical procedures throughout the software life cycle to identify, define, and baseline software items, including documentation, in a system; control modifications and releases of the items; and record and report the status of the items and modification requests. Configuration management is necessary to recreate a software item, to identify its constituent parts, and to provide a history of the changes that have been made to it.

A.6.3.1 Process implementation

This activity requires the developer or maintainer to plan configuration management. Specifying configuration management responsibilities and procedures is necessary to control software items and documentation during both development and maintenance. The plan may specify different procedures in various stages of the life cycle or different procedures for various stages of development of a particular item, with more rigorous procedures applying in later stages.

A.6.3.2 Configuration identification

This activity requires the developer/maintainer to uniquely identify software configuration items and their versions. This identification is necessary to identify the software configuration items and the versions that are included in the medical device software.

A.6.3.3 Configuration control

This activity requires the developer/maintainer to control modification of the software configuration items and to record information identifying change requests and providing documentation about their disposition. This activity is necessary to ensure that unauthorized or unintended changes are not made to the software configuration items and to ensure that approved changes are implemented. Change requests are often approved by a change control board or by a manager or technical lead according to the configuration management plan. The approved change request is then linked in some way with the actual modification to the software. This linkage may be accomplished with a software configuration management tool or by maintaining a matrix showing traceability from the change request to the software that was changed. The requirement is that each modification be linked to a change request and that documentation exists to show that the change request was approved. The documentation may be change control board minutes, an approval signature, or a record in a database.

A.6.3.4 Configuration status accounting

This activity requires the developer/maintainer to maintain records of the history of the software configuration items. This activity is necessary to determine when and why changes were made. Access to this information is necessary to ensure that software configuration items contain only authorized modifications.

A.6.4 Verification process

Software verification is the confirmation by examination and provision of objective evidence that the output of a particular phase of software development meets all of the input requirements for that phase. The verification process is a process for determining whether the software product of an activity fulfills the requirements or conditions imposed on it in the previous activities. Verification is not simply testing. Testing cannot show the absence of errors. The purpose of verification is to ensure the correctness of the product. Verification may include analysis, review, and testing. Verification detects and reports defects that have been introduced during the software development process.

Peer technical review is a powerful technique that is often used for verification. There are many variations of technical reviews, but the basic process is that a work product that is free of known defects is reviewed by a group of peers of the author of the work product. Any defects are documented to be addressed by the author after completion of the technical review.

A.6.4.1 Process implementation

This activity requires the developer to establish a verification process, select someone to conduct it, develop and implement plans for verification, and resolve nonconformances. Verification detects defects made as a development activity translates an input to the output of that activity. It determines whether each deliverable follows correctly from its predecessor. During a development, there must be a series of verification activities looking for defects made during each step of the development process. Removing defects as soon as possible after they are made is essential in producing high quality software.

A.6.4.2 Verification

The intent of this verification activity is to provide assurance that the software life cycle, as defined under 5.1.1.1, was implemented and adhered to. For the life cycle to meet the requirements of this standard, its phases that have meaningful deliverables and observable starting and ending points must be defined. This standard does not prescribe specific phases or phase deliverables, and no specific life cycle (e.g., waterfall) is required. Phases may be iterative or overlapping.

One approach to meeting this verification requirement is to define formal phase transition requirements, or even to hold phase transition reviews, and document the dates of each transition. Phase transition requirements are not restricted to documents. Phase transition requirements can include draft or final documents, prototypes or build milestones, simulation or test results, specific reviews or approvals, or other items.

A.6.4.2.2 Software requirements verification

This activity requires the developer to verify the outputs of the software requirements analysis activity.

A.6.4.2.4 Software detailed design verification

This activity requires the developer to verify the outputs of both the architecture and the detailed design activities. The design specifies how the requirements are to be implemented. If the design contains defects, the code will not implement the requirements correctly.

A.6.4.2.5 Software coding and testing verification

This activity requires the developer to verify the code. If the code does not implement the design correctly, the medical device software will not perform as intended.

A.6.4.2.6 Software integration and testing verification

This activity requires the developer to verify the output of the software integration and testing activity. The output of the software integration and testing activity is the integrated software items. These integrated software items must function properly for the entire medical device software to function correctly and safely.

A.6.4.2.7 Software system testing verification

The results of the software system testing must be reviewed to ensure that the expected results were obtained.

A.6.4.2.8 Documentation verification

This activity requires the developer to verify the outputs of the documentation process. The documentation being produced must be complete, consistent, and accurate.

A.6.5 Problem resolution process

The problem resolution process is a process for analyzing and resolving the problems (including nonconformances), whatever their nature or source, including those discovered during the execution of development, maintenance, or other processes. The objective is to provide a timely, responsible, and documented means to ensure that discovered problems are analyzed and resolved and that trends are recognized. This process is sometimes called “defect tracking” in software engineering literature. It is called “problem resolution” in ISO/IEC 12207 and “problem reporting” in IEC 60601-1-4, Amendment A. We have chosen to call it “problem resolution” in this standard.

A.6.5.1 Process implementation

This activity requires that the developer establish a process for handling problems and specify when that process must be used. Tracking problems and documenting their disposition are essential activities in maintaining control over the changes being made to the software. It is not required that all problems must be corrected. However, all problems must be evaluated for their possible relevance to safety as specified in ANSI/AAMI/ISO 14971. If the problem is not corrected, the risk evaluation must be reviewed and updated if necessary.

A.6.5.2 Problem resolution

This activity requires that the developer use the problem resolution process when a problem or nonconformance is identified.

Annex B
(informative)

References

IEC 60601-1-4:1996, *Medical electrical equipment—Part 1: General requirements for safety. Section 1.4 Collateral standard: General requirements for programmable electrical medical systems.*

IEC 61508:1998, *Functional safety of electrical/electronic/programmable electronic safety-related systems.*

IEEE 610.12:1990, *IEEE standard glossary of software engineering terminology.*

ISO 9000-3:1997, *Quality management and quality assurance standards—Part 3: Guidelines for the application of ISO 9001:1994 to the development, supply, installation, and maintenance of computer software.*

ISO 9000:2000, *Quality management systems—Fundamentals and vocabulary.*

ISO 9001:1994, *Quality systems—Models for quality assurance in design, development, production, installation, and servicing.*

ISO/IEC 12207:1995, *Information technology—Software life cycle processes.*

ISO/IEC Guide 51:1999, *Safety aspects—Guidelines for their inclusion in standards.*

