

Technical Information Report

AAMI TIR32:2004/ (R)2016

Medical device software
risk management

Medical device software risk management

Approved 23 December 2004 and reaffirmed 26 May 2016 by
Association for the Advancement of Medical Instrumentation

Abstract: This AAMI technical information report provides information useful to performing effective software risk management, a significant part of the overall risk management process for medical devices containing software. It does this in the context of ANSI/AAMI/ISO 14971:2000, *Medical devices—Application of risk management to medical devices*, and in the context of ANSI/AAMI SW68:2001, *Medical device software—Software life cycle processes*.

Keywords: hazard, risk, software

AAMI Technical Information Report

A technical information report (TIR) is a publication of the Association for the Advancement of Medical Instrumentation (AAMI) Standards Board that addresses a particular aspect of medical technology.

Although the material presented in a TIR may need further evaluation by experts, releasing the information is valuable because the industry and the professions have an immediate need for it.

A TIR differs markedly from a standard or recommended practice, and readers should understand the differences between these documents.

Standards and recommended practices are subject to a formal process of committee approval, public review, and resolution of all comments. This process of consensus is supervised by the AAMI Standards Board and, in the case of American National Standards, by the American National Standards Institute.

A TIR is not subject to the same formal approval process as a standard. However, a TIR is approved for distribution by a technical committee and the AAMI Standards Board.

Another difference is that, although both standards and TIRs are periodically reviewed, a standard must be acted on—reaffirmed, revised, or withdrawn—and the action formally approved usually every five years but at least every 10 years. For a TIR, AAMI consults with a technical committee about five years after the publication date (and periodically thereafter) for guidance on whether the document is still useful—that is, to check that the information is relevant or of historical value. If the information is not useful, the TIR is removed from circulation.

A TIR may be developed because it is more responsive to underlying safety or performance issues than a standard or recommended practice, or because achieving consensus is extremely difficult or unlikely. Unlike a standard, a TIR permits the inclusion of differing viewpoints on technical issues.

CAUTION NOTICE: This AAMI TIR may be revised or withdrawn at any time. Because it addresses a rapidly evolving field or technology, readers are cautioned to ensure that they have also considered information that may be more recent than this document.

All 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.

Comments on this technical information report are invited and should be sent to AAMI, Attn: Standards Department, 1110 N. Glebe Road, Suite 220, Arlington, VA 22201-4795.

Published by

Association for the Advancement of Medical Instrumentation
4301 N. Fairfax Drive, Suite 301
Arlington, VA 22203-1633
www.aami.org

© 2005 by the Association for the Advancement of Medical Instrumentation

All Rights Reserved

Publication, reproduction, photocopying, storage, or transmission, electronically or otherwise, of all or any part of this document without the prior written permission of the Association for the Advancement of Medical Instrumentation is strictly prohibited by law. It is illegal under federal law (17 U.S.C. § 101, *et seq.*) to make copies of all or any part of this document (whether internally or externally) without the prior written permission of the Association for the Advancement of Medical Instrumentation. Violators risk legal action, including civil and criminal penalties, and damages of \$100,000 per offense. For permission regarding the use of all or any part of this document, contact AAMI at 4301 N. Fairfax Drive, Suite 301, Arlington, VA 22203-1633. Phone: (703) 525-4890; Fax: (703) 525-1067.

Printed in the United States of America

ISBN 1-57020-233-8

Contents

	Page
Glossary of equivalent standards	v
Committee representation	vii
Foreword	ix
Introduction	x
1 Scope.....	1
1.1 Purpose.....	1
1.2 Field of application	1
1.3 Usage.....	2
1.4 Organization.....	2
1.5 Limitations	2
2 References.....	2
3 Definitions	3
4 Perspective 1: Basic concepts of medical device software risk management.....	6
4.1 Medical device risk management.....	6
4.1.1 Software risk cannot be managed effectively in isolation	6
4.1.2 Software input is an important part of device risk management.....	7
4.2 Software risk management.....	7
4.3 Risk control	11
4.3.1 Risk control through development assurance levels	14
4.3.2 Achieving development assurance level requirements	14
4.4 Integration of risk management in the software life cycle	15
4.4.1 Risk management is also essential for software maintenance	16
4.5 Common confusion regarding software risk management	16
5 Perspective 2: Software considerations in medical device risk management.....	17
5.1 Risk analysis	17
5.1.1 Risk analysis procedure.....	18
5.1.2 Intended-use or intended-purpose identification	19
5.1.3 Identification of known or foreseeable hazards	20
5.1.4 Estimation of the risks for each hazard	23
5.2 Risk evaluation.....	25
5.3 Risk control	27
5.3.1 Options analysis.....	27
5.3.2 Implementation of risk control measures	29
5.3.3 Residual risk evaluation	29
5.3.4 Other generated hazards	30
5.4 Risk management report.....	30
5.5 Postproduction information.....	31
6 Perspective 3: Software risk management within a software life cycle.....	32
6.1 Risk management–life cycle integration	32
6.2 ANSI/AAMI SW68:2001 development process	35
6.2.1 Process implementation.....	35
6.2.2 Software requirements analysis	36
6.2.3 Software architectural design.....	38
6.2.4 Software detailed design.....	41
6.2.5 Code and unit test.....	42
6.2.6 Integration, system, and validation testing	46
6.2.7 Software release	48

6.3	ANSI/AAMI SW68 maintenance process	49
6.3.1	Process implementation.....	50
6.3.2	Problem and modification analysis and implementation.....	50
7	Perspective 4: Soft factors in software risk management.....	52
7.1	Intended-use and domain knowledge.....	52
7.2	Team dynamics.....	53
7.3	Management	53
7.4	Programming experience and attitude.....	53
7.5	Technical knowledge.....	54

Annexes

A	Direct causes sample table	55
B	Indirect causes and risk control measures table (failures due to unpredictable behaviors).....	62

Tables

1	Examples of risk control measures	27
2	Life cycle–risk management grid.....	33
3	Types of analyses	37
4	Various methods of software redundancy	40
5	Methods to facilitate assurance that risk control methods are likely to perform as intended	47

Figures

1	Hazard–cause continuum	5
2	Software risk management context diagram	8
3	Types of functionality	9
4	Causal chains	10
5	First and last points of control context diagram.....	12
6	First and last points of software control in causal chains	13
7	Direct and indirect causes.....	22

Glossary of equivalent standards

International Standards adopted in the United States may include normative references to other International Standards. For each International Standard that has been adopted by AAMI (and ANSI), the table below gives the corresponding U.S. designation and level of equivalency to the International Standard.

NOTE—Documents are sorted by international designation.

Other normatively referenced International Standards may be under consideration for U.S. adoption by AAMI; therefore, this list should not be considered exhaustive.

International designation	U.S. designation	Equivalency
IEC 60601-1-2:2001 and Amendment 1:2004	ANSI/AAMI/IEC 60601-1-2:2001 and Amendment 1:2004	Identical
IEC 60601-2-04:2002	ANSI/AAMI DF80:2003	Major technical variations
IEC 60601-2-19:1990 and Amendment 1:1996	ANSI/AAMI I136:2004	Major technical variations
IEC 60601-2-20:1990 and Amendment 1:1996	ANSI/AAMI I151:2004	Major technical variations
IEC 60601-2-21:1994 and Amendment 1:1996	ANSI/AAMI/IEC 60601-2-21 and Amendment 1:2000 (consolidated texts)	Identical
IEC 60601-2-24:1998	ANSI/AAMI ID26:2004	Major technical variations
IEC TR 60878:2003	ANSI/AAMI/IEC TIR60878:2003	Identical
IEC TR 62296:2003	ANSI/AAMI/IEC TIR62296:2003	Identical
ISO 5840:200x ¹	ANSI/AAMI/ISO 5840:2005	Identical
ISO 7198:1998	ANSI/AAMI/ISO 7198:1998/2001/(R)2004	Identical
ISO 7199:1996	ANSI/AAMI/ISO 7199:1996/(R)2002	Identical
ISO 10993-1:2003	ANSI/AAMI/ISO 10993-1:2003	Identical
ISO 10993-2:1992	ANSI/AAMI/ISO 10993-2:1993/(R)2001	Identical
ISO 10993-3:2003	ANSI/AAMI/ISO 10993-3:2003	Identical
ISO 10993-4:2002	ANSI/AAMI/ISO 10993-4:2002	Identical
ISO 10993-5:1999	ANSI/AAMI/ISO 10993-5:1999	Identical
ISO 10993-6:1994	ANSI/AAMI/ISO 10993-6:1995/(R)2001	Identical
ISO 10993-7:1995	ANSI/AAMI/ISO 10993-7:1995/(R)2001	Identical
ISO 10993-9:1999	ANSI/AAMI/ISO 10993-9:1999	Identical
ISO 10993-10:2002	ANSI/AAMI BE78:2002	Minor technical variations
ISO 10993-11:1993	ANSI/AAMI 10993-11:1993	Minor technical variations
ISO 10993-12:2002	ANSI/AAMI/ISO 10993-12:2002	Identical
ISO 10993-13:1998	ANSI/AAMI/ISO 10993-13:1999/(R)2004	Identical
ISO 10993-14:2001	ANSI/AAMI/ISO 10993-14:2001	Identical
ISO 10993-15:2000	ANSI/AAMI/ISO 10993-15:2000	Identical

¹ Currently at FDIS stage

International designation	U.S. designation	Equivalency
ISO 10993-16:1997	ANSI/AAMI/ISO 10993-16:1997/(R)2003	Identical
ISO 10993-17:2002	ANSI/AAMI/ISO 10993-17:2002	Identical
ISO 11134:1994	ANSI/AAMI/ISO 11134:1993	Identical
ISO 11135:1994	ANSI/AAMI/ISO 11135:1994	Identical
ISO 11137:1995 and Amdt 1:2001	ANSI/AAMI/ISO 11137:1994 and A1:2002	Identical
ISO 11138-1:1994	ANSI/AAMI ST59:1999	Major technical variations
ISO 11138-2:1994	ANSI/AAMI ST21:1999	Major technical variations
ISO 11138-3:1995	ANSI/AAMI ST19:1999	Major technical variations
ISO TS 11139:2001	ANSI/AAMI/ISO 11139:2002	Identical
ISO 11140-1:1995 and Technical Corrigendum 1:1998	ANSI/AAMI ST60:1996	Major technical variations
ISO 11140-5:2000	ANSI/AAMI ST66:1999	Major technical variations
ISO 11607:2003	ANSI/AAMI/ISO 11607:2000	Identical
ISO 11737-1:1995	ANSI/AAMI/ISO 11737-1:1995	Identical
ISO 11737-2:1998	ANSI/AAMI/ISO 11737-2:1998	Identical
ISO 11737-3:2004	ANSI/AAMI/ISO 11737-3:2004	Identical
ISO TR 13409:1996	AAMI/ISO TIR13409:1996	Identical
ISO 13485:2003	ANSI/AAMI/ISO 13485:2003	Identical
ISO 13488:1996	ANSI/AAMI/ISO 13488:1996	Identical
ISO 14155-1:2003	ANSI/AAMI/ISO 14155-1:2003	Identical
ISO 14155-2:2003	ANSI/AAMI/ISO 14155-2:2003	Identical
ISO 14160:1998	ANSI/AAMI/ISO 14160:1998	Identical
ISO 14161:2000	ANSI/AAMI/ISO 14161:2000	Identical
ISO 14937:2000	ANSI/AAMI/ISO 14937:2000	Identical
ISO TR 14969:2004	ANSI/AAMI/ISO TIR14969:2004	Identical
ISO 14971:2000 and A1:2003	ANSI/AAMI/ISO 14971:2000 and A1:2003	Identical
ISO 15223:2000, A1:2002, and A2:2004	ANSI/AAMI/ISO 15223:2000, A1:2001, and A2:2004	Identical
ISO 15225:2000 and A1:2004	ANSI/AAMI/ISO 15225:2000 and A1:2004	Identical
ISO 15674:2001	ANSI/AAMI/ISO 15674:2001	Identical
ISO 15675:2001	ANSI/AAMI/ISO 15675:2001	Identical
ISO TS 15843:2000	ANSI/AAMI/ISO TIR15843:2000	Identical
ISO TR 15844:1998	AAMI/ISO TIR15844:1998	Identical
ISO 15882:2003	ANSI/AAMI/ISO 15882:2003	Identical
ISO TR 16142:1999	ANSI/AAMI/ISO TIR16142:2000	Identical
ISO 17664:2004	ANSI/AAMI ST81:2004	Major technical variations
ISO 25539-1:2003	ANSI/AAMI/ISO 25539-1:2003	Identical

Committee representation

Association for the Advancement of Medical Instrumentation

AAMI Medical Device Software Committee

This technical information report was developed by the Software Risk Management Task Group of the AAMI Medical Device Software Committee. Approval of this TIR 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:

<i>Cochairs:</i>	Sherman Eagles John F. Murray, Jr.
<i>Secretary:</i>	Nancy George
<i>Members:</i>	Robert G. Britain, National Electrical Manufacturers Association (NEMA) Warren P. Dickinson, Ion Beam Applications Sherman Eagles, Medtronic Inc. Christine M. Flahive, Christine M. Flahive Associates John J. Flynn, Hill-Rom Company Richard C. Fries, Instrumentarium USA Inc. Larry A. Fry, Draeger Medical Nancy George, Software Quality Management Inc. Larry Gillum, Cardinal Health Medical Products and Services Group Steven Gitelis, Guidant Corp. Lori Haller, STERIS Corp. James P. Hempel, Tyco Healthcare/US Surgical Neil Holland, Abbott Laboratories David R. Jones, Philips Medical Systems Christopher Keegan, Welch Allyn Inc. Charlie D. King, Siemens Medical Systems Alan Kusnitz, SoftwareCPR Bernie Liebler, Advanced Medical Technology Association (AdvaMed) Mary Beth McDonald, St. Jude Medical E. Paul Morozoff, Spacelabs Medical Inc. Paul Mueller, Bausch & Lomb Inc. John F. Murray, Jr., U.S. Food and Drug Administration Harvey Rudolph, PhD, Underwriters Laboratories Inc. Carla Sivak, Mitek/Johnson & Johnson Christine Strysik, Baxter Healthcare Corp. Steven D. Walter, Becton Dickinson
<i>Alternates:</i>	Gregory Whitney, CR Bard Christopher P. Clark, Bausch & Lomb Inc. Brian J. Fitzgerald, U.S. Food and Drug Administration Christopher D. Ganser, CR Bard Florentino Kimpo, Medtronic Physio-Control Gretel Lumley, Philips Medical Systems David McCall, STERIS Corp. Dennis Mertz, Becton Dickinson Carl Pantiskas, Spacelabs Medical Inc. Raj G. Raghavendran, Ethicon Endo-Surgery/Johnson & Johnson Robert Smith, St. Jude Medical Fayez Sweiss, Abbott Laboratories Donna Bea Tillman, PhD, U.S. Food and Drug Administration Stephen Vastagh, National Electrical Manufacturers Association (NEMA)

The AAMI Medical Device Software Committee gratefully acknowledges the work of its **Software Risk Management Task Group**. At the time this document was published, the task group had the following members:

Cochairs: Paul L. Jones
Alan Kusnitz
Secretary: Annette M. Hillring
Members: Sherman Eagles, Medtronic Inc.
Lucille Ferus, SoftwareCPR
Stan Hamilton, SoftwareCPR
Annette M. Hillring, Johnson & Johnson
Paul L. Jones, U.S. Food and Drug Administration
Alan Kusnitz, SoftwareCPR
Eric Linner, Baxter Healthcare Corp.
John F. Murray, Jr., U.S. Food and Drug Administration
Brian Pate, GE Healthcare
Raj G. Raghavendran, Ethicon Endo-Surgery/Johnson & Johnson
Yves Theriault, STERIS Corp.

The AAMI Medical Device Software Committee and its Software Risk Management Task Group would like to thank the following individuals for their invaluable contribution to the development of this TIR:

Sean M. Beatty, High Impact Services Inc.
Doug Lichorwic, Northrup Grumman
David Vogel, Intertech Engineering Associates, Inc.

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

Foreword

Effective software risk management is a significant part of the overall risk management process for medical devices containing software. This technical information report (TIR) provides information useful to performing effective software risk management. It does this in the context of ANSI/AAMI/ISO 14971:2000, *Medical devices—Application of risk management to medical devices*, and in the context of ANSI/AAMI SW68:2001, *Medical device software—Software life cycle processes*.

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 implementation of the software fulfills those intentions without causing any unacceptable risks.

Accidents are often preceded by a belief that they cannot happen. People often believe that software is designed to work properly and that testing ensures that it will work properly, despite a general recognition that neither quality nor safety can be “tested into” software. The fact is that most software testing does little more than exercise a small sampling of the software logic in all but the simplest of programs.

Ignoring the possibility of defects in software can lead to the release of medical device software that can fail in ways that affect device safety and effectiveness. This report is predicated on the notion that, as part of the software development process and general device risk management process, specific focus is required on

- identifying software’s relationship to potential device hazards, both in terms of intended software functionality and the effects of potential software defects;
- identifying adequate risk control measures in terms of software and nonsoftware design; and
- verifying the implementation and effectiveness of risk control measures.

The cost of diligence in this regard is inconsequential compared to the cost of an accident in any way one would like to define cost (e.g., human, financial, legal, or regulatory exposure).

Many standards have taken the approach of having separate “safety” and “performance” standards for medical electrical equipment. This approach was a natural extension of the historical approach taken at the national and international level with other electrical equipment standards (e.g., those for consumer electronics), where basic physical safety is regulated through mandatory standards but other “performance” specifications are regulated by market pressure. In this context, one could say, “The ability of an electric kettle to boil water is not critical to its safe use!”

This is not the situation with medical devices. Responsible organizations must depend on standards to ensure effectiveness as well as basic safety. The accuracy with which the equipment controls the delivery of energy or therapeutic substances to the patient is of concern because a lack of effectiveness can become a safety issue. Likewise, the manner in which medical device software processes and displays physiological or diagnostic data is of concern because it can affect patient management. Medical authorities are equally concerned about the ability of the equipment to prevent hazards and to perform clinical functions effectively. An increasing amount of the clinical functionality of many medical devices is controlled by a software subsystem of the medical device.

Thus, it is sometimes difficult to make clear distinctions between safety and effectiveness. As such, where this report uses the term “safety,” it is intended to include effectiveness in cases where the lack of effectiveness can be a safety issue.

The benefit derived by the patient must always be considered along with the risk in using a medical device. This implies that the need for protection from risk caused by the medical device differs depending on the risk to the patient of not receiving treatment, and these varying needs must be considered as part of the overall risk evaluation of the device.

Many standards and publications—national, international, and sector specific—address risk management, and others address software life cycle development processes. However, none of these documents address software system safety design issues in the context of medical device systems. Additionally, software-related aspects of medical device risk management need to be explained in software terms with software examples. The AAMI Medical Device Software Committee recognizes a need for this information and is addressing the issue by way of this technical information report.

Medical device software risk management

1 Scope

This technical information report (TIR) should be regarded as a reference for developing safe software systems to be used in medical devices. The information that it contains provides a framework within which experience, insight, and judgment are applied systematically to reduce medical device risks. The TIR does this in the context of ANSI/AAMI/ISO 14971:2000, *Medical devices—Application of risk management to medical devices*, and in the context of ANSI/AAMI SW68:2001, *Medical device software—Software life cycle processes*.

For readers to understand the scope of this document, it is important to understand the distinction between software safety and software reliability. The National Institute of Science and Technology information report [NISTIR 5589] on software hazard analysis states this distinction quite clearly:

Software safety should not be confused with software reliability. Reliability is the ability of a system to perform its required functions under stated conditions for a specified period of time [IEEE610]. Safety is the probability that conditions (hazards) that can lead to a mishap do not occur, whether or not the intended function is performed [LEVESON86]. Reliability is interested in all possible software errors, while safety is concerned only with those errors that cause actual system hazards [LEVESON86]. . . . Software safety and software reliability are part of software quality. Quality is the degree to which a system meets specified requirements, and customer or user needs or expectations [IEEE610].

Many of the same techniques used to ensure software reliability and quality are relevant for ensuring software safety. This report does not discuss general aspects of software quality assurance.

1.1 Purpose

The goal of this TIR is to be a technical reference on risk management for medical devices. It is intended primarily for software engineers, software quality assurance personnel, and those responsible for medical device risk management. Others involved in medical device product development, quality assurance, regulatory affairs, and auditing may also find this document useful.

The report attempts to clarify process relationships outlined in ANSI/AAMI SW68:2001, *Medical device software—Software life cycle processes*, and ANSI/AAMI/ISO 14971:2000, *Medical devices—Application of risk management to medical devices*, in the context of software system safety, keeping in mind the varied interests of the audience.

Understanding the terminology and its proper context is key to understanding the associated processes. This report attempts to clarify some of those subtleties by looking at the components of risk management, and by using precise language to identify how those components relate to each other. For example, ISO definitions such as *hazard*, *harm*, and *safety* are clarified through use of additional terms and examples from a software perspective.

The report provides guidance for those new to the concepts of software system safety in the medical device industry and as an aide-memoire for medical device and software designers more familiar with the topic.

The first objective of this report is to provide those working “down in the trenches” with some insight into safety considerations when using software in a medical device. A second objective is to help risk managers understand the implications for risk management posed by the presence of software in the system. All too often, those charged with the responsibility for developing software and those charged with the responsibility of managing risk operate independently of each other. It is a goal of this report to help bridge this divide by fostering communication and a shared understanding of the relationship between software engineering and risk management.

1.2 Field of application

This TIR contains information applicable to risk management for the entire array of medical device software, including:

- Embedded software systems (e.g., glucose meter firmware)
- Stand-alone software systems (e.g., dosage calculations programs)

- Information systems (e.g., clinical information systems)
- Accessory systems (e.g., radiation planning systems)
- Telemedicine systems (e.g., remote imaging systems)

1.3 Usage

This report discusses medical device risk management in the context of software. The report is technical in nature and does not dwell on processes specified in other relevant consensus standards, ANSI/AAMI/ISO 14971:2000 and ANSI/AAMI SW68:2001 in particular. The expectation is that software engineers will use the principles of software risk management described in this report together with other general risk management and software engineering standards and reference and educational material.

1.4 Organization

This report is organized into four major sections that contribute to the subject matter from a different perspective:

- Perspective 1: Basic concepts of medical device software risk management (section 4)

This section presents key concepts and terms useful for ensuring software safety and effectiveness.

- Perspective 2: Software considerations in medical device risk management (section 5)

This section goes through major elements of the medical device risk management process defined in ANSI/AAMI/ISO 14971:2000 and provides a software perspective for each risk management activity.

- Perspective 3: Software risk management within a software life cycle (section 6)

This section steps through the activities of the two primary software life cycle processes defined in ANSI/AAMI SW68:2001 (development and maintenance) and discusses risk management considerations for each activity of the software life cycle.

- Perspective 4: Soft factors in software risk management (section 7)

This section identifies a number of soft factors that are important to generate safe and effective software.

Two important annexes (Annexes A and B) are referenced from several sections. These annexes contain tables that provide examples of typical medical device software functionality related to safety, of potential software failures that could contribute to hazardous situations, and of potential risk control measures corresponding to these failures.

Since risk management is addressed from different perspectives in each major section, there is some intentional redundancy in the information provided.

1.5 Limitations

This report is not a complete exposition of all of the software engineering tools and techniques used to manage risk. It discusses approaches to reduce risk but does not define specific acceptable risk levels. It was not developed to explicitly address software used in medical device manufacturing, quality systems, design, development, or electronic recordkeeping, although the concepts presented could be useful in some respects.

2 References

ANSI/AAMI SW68:2001, *Medical device software—Software life cycle processes*.

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

DEF STAN 00-55, 00-56:1997, *Requirements for Safety-Related Equipment in Defence Equipment*. Ministry of Defence, Directorate of Standardization, Glasgow, Scotland.

IEC 60513:1994, *Fundamental aspects of safety standards for medical electrical equipment*.

IEC 61508-3:1998, *Functional safety of electrical/electronic/programmable electronic safety-related systems—Part 3: Software requirements*.

IEEE 610:1990, *IEEE computer dictionary—Compilation of IEEE standard computer glossaries*.

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

ISO/IEC 12207:1995/Amendment 1:2002, *Information technology—Software life cycle processes*.

ISO/IEC 15026:1998, *Information technology—System and software integrity levels*.

ISO/IEC 60601-1-4:2000, *Medical electrical equipment—Part 4: Programmable electrical medical systems*.

NISTIR 5589:1995, *A study on hazard analysis in high-integrity software standards and guidelines*.

Pradhan, DK, *Fault-Tolerant Computer System Design*. Upper Saddle River (NJ): Prentice-Hall, 1996.

3 Definitions

3.1 corrective measures: Risk control measures meant to detect a potential cause of a hazardous situation and reduce the severity or probability of harm occurring.

3.2 direct cause: Defect whose results are predictable and that can contribute to one or more hazards.

3.3 diversity: Characteristic of risk control measures that uses different architectures, components, algorithms, development processes, or other techniques to reduce the risk of a single software defect resulting in a hazard.

3.4 failsafe: Management of failure modes to prevent harm; performance characteristics necessary to maintain risk within acceptable limits. (See IEC 60601-1-4:2000.)

3.5 harm: Physical injury or damage to the health of people, damage to property, or damage to the environment. (See ANSI/AAMI/ISO 14971:2000.)

3.6 hazard: Potential source of harm. (See ANSI/AAMI/ISO 14971:2000.)

3.7 hazardous event: Any occurrence of a hazard. (See ISO/IEC Guide 51:1999.) A hazardous situation that results in harm.

3.8 hazardous situation: Circumstance in which people, property, or the environment are exposed to one or more hazards. (See ANSI/AAMI/ISO 14971:2000.)

3.9 indirect cause: Defect that could have unpredictable side effects and result in one or more hazards (e.g., a memory overflow that could corrupt safety-related code or data).

3.10 intended use/intended purpose: Use of a product, process, or service in accordance with the specifications, instructions, and information provided by the manufacturer. (See ANSI/AAMI/ISO 14971:2000.)

3.11 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. (See ISO/IEC 12207:1995.)

3.12 misuse: Any use of a device, whether malicious or not, that is not within the parameters of the intended use.

3.13 mitigative measures: Risk control measures meant to reduce the severity of a hazardous situation.

3.14 preventive measures: Risk control measures that prevent causes from contributing to hazardous situations.

3.15 protective measures: Actions taken to reduce the risk of hazards and ultimately the risk of harm. These actions may be preventive, corrective, or mitigative. Although the term *mitigation* is sometimes used as a synonym for protective measure, *mitigation* has a more specific meaning as indicated above. *Protective measures* or *risk control measures* are the generic terms used to describe the steps taken to reduce risk.

3.16 reliability: Predictable and consistent performance of the software under conditions specified in the design basis. This top-level attribute is important to safety because it decreases the likelihood that faults causing unsuccessful operation will be introduced into the source code during implementation. (See IEEE 610:1990.)

3.17 residual risk: Risk remaining after protective measures have been taken. (See ANSI/AAMI/ISO 14971:2000.)

3.18 risk: Combination of the probability of occurrence of harm and the severity of that harm. (See ANSI/AAMI/ISO 14971:2000.)

3.19 risk analysis: Systematic use of available information to identify hazards and to estimate the risk. (See ANSI/AAMI/ISO 14971:2000.)

3.20 risk assessment: Overall process comprising a risk analysis and a risk evaluation. (See ANSI/AAMI/ISO 14971:2000.)

3.21 risk control: Process through which decisions are reached and protective measures are implemented for reducing risks to, or maintaining risks within, specified levels. (See ANSI/AAMI/ISO 14971:2000.)

3.22 risk control measure: Methods used to reduce the severity or probability of harm.

3.23 risk evaluation: Judgment, on the basis of risk analysis, of whether a risk that is acceptable has been achieved in a given context, given the current values of society. (See ANSI/AAMI/ISO 14971:2000.)

3.24 risk management: Systematic application of management policies, procedures, and practices to the tasks of analyzing, evaluating, and controlling risk. (See ANSI/AAMI/ISO 14971:2000.)

3.25 robustness: Capability of the safety system software to operate in an acceptable manner under abnormal conditions or events. This top-level attribute is important to safety because it enhances the capability of the software to handle exception conditions, recover from internal failures, and prevent propagation of errors arising from unusual circumstances (not all of which may have been fully defined in the design basis). (See IEEE 610:1990.)

3.26 safety-related software: Software that, if defective, could result in a hazardous situation.

3.27 safe state: Device modes that are free from unacceptable risk.

3.28 safety: Freedom from unacceptable risk. (See ANSI/AAMI/ISO 14971:2000.)

3.29 safety-related requirements: Requirements that, if not met, could result in a hazardous situation.

3.30 severity: Measure of the possible consequences of a hazard. (See ANSI/AAMI/ISO 14971:2000.)

3.31 software item: Any identifiable part of a software product. (See ISO 9000-3:1997.)

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

3.33 system: Term used in software standards such as ISO/IEC 12207:1995 to include the computer hardware on which the software will run. In general, *system* refers to the entire system and not just to software, electronics, mechanics, fluidics, or other specific components. In the context of this report, the term *medical device* is used to refer to the system of which software is a part.

3.34 systematic failure: Design errors that do not occur randomly and for which a probability of occurrence cannot be calculated. (See ANSI/AAMI/ISO 14971:2000, Annex E 4.²)

Discussion of definitions

In the ANSI/AAMI/ISO 14971:2000 taxonomy, there is *harm*, with an emphasis on clinical mishap and effect; there are *hazards*, which represent potential sources of harm; and there are *causes* or *contributing factors* for the hazards.

This definition of *hazard* contributes to a great deal of confusion when compiling a list of known or foreseeable hazards. This confusion arises, in part, because the definition is ambiguous: almost every state or event within the device system, given certain conditions, could be construed as a potential source of harm. The distinction, then, between a hazard and the events resulting in a hazard can become very subjective. For example, if the device has a component insulation failure, resulting in an electrical short, resulting in an electrical shock, resulting in cardiac arrhythmia, resulting in fibrillation or death, which of these events is the potential source of *harm* (i.e., the *hazard*), and which is the cause of the hazard? Similarly, if a device software component performs an incorrect calculation, resulting in an incorrect electrical stimulation, resulting in cardiac arrhythmia, which event is the potential source of harm, and which is the cause of the hazard? The ambiguity of the definition results in hazard lists that often vary widely between manufacturers of similar devices.

What constitutes a hazard depends on how the designers abstract the boundaries of the device or system, as shown in Figure 1. It can also depend on the intended use of the device, and even how the designers define a *hazard*. It would be useful to regulators, auditors, and manufacturers if these boundaries were unambiguously standardized. ANSI/AAMI/ISO 14971:2000 begins to address this issue by providing lists of clinical hazards and classes of hazardous conditions in its annexes.

² A random failure is associated with something that works but over time fails because of lifetime issues. Software does not wear out. It fails under the same conditions whenever they occur.

The diagram of Figure 1 provides a model of the medical device *hazard–cause* continuum. The device may affect the patient, user, and service domains by means of direct or indirect (environmental) interfaces in such a manner as to cause harm. Clinical and device-level hazards listed in the diagram generally map to the ANSI/AAMI/ISO 14971:2000 annexes and may serve as a starting point for establishing hazards.

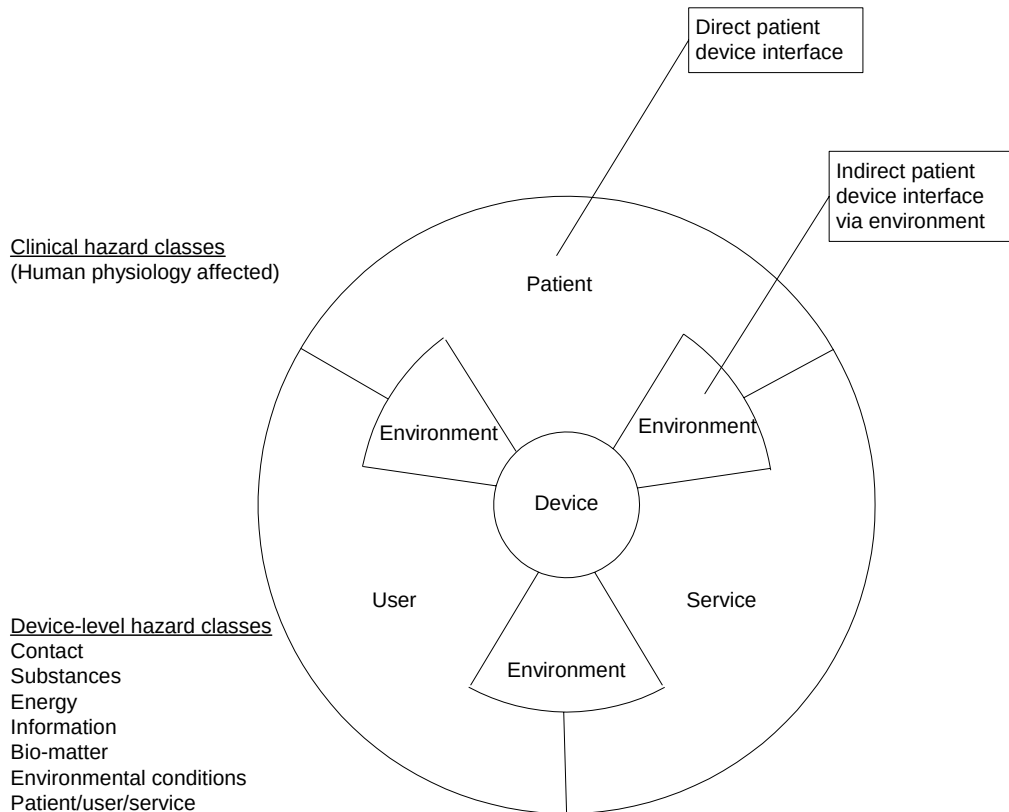


Figure 1—Hazard cause continuum

In this report, an attempt is made to use the terms in a less ambiguous manner. It is believed that these refinements are consistent with the spirit of ISO/IEC definitions and will help resolve some of the confusion experienced by medical device developers.

The term *hazard*, therefore, is used to refer to any broadly characterized means, mode, or manner by which a medical device (or, more generally, a system being analyzed) might cause harm. The classification of hazards is highly subjective and purely pragmatic. The fundamental test for classifying hazards is whether the proposed classification scheme provides insights that are useful for analyzing risk. For example, many hazards involve exposure to harmful amounts of energy, provision of incorrect diagnostic information, or provision of incorrect therapy.

A *hazardous event* is any occurrence of a hazard. Whether harm results from a given hazardous event depends on the degree to which hazardous preconditions are present, the precise timing of contributory events, and the effectiveness of any risk control measures designed into the system.

A cause of a hazard is any set of events or circumstances, the combination of which might reasonably be expected to result in a hazardous event. A given hazard might have one, several, or many possible causes (*contributing factors*). Additionally, hazards might be caused by logical combinations of precursor causes or, indirectly, by random failures.

If we apply these definitions to the example cited earlier, electrical shock is most properly identified as the hazard. The insulation failure, together with the circumstance (e.g., shock, vibration, or handling) that caused the exposed conductors to become shorted together, constitutes a possible cause of the hazard. One might also identify other entirely different causes of electrical shock (misassembled components, software algorithm error, operator error, etc.).

The medical conditions of cardiac arrhythmia and fibrillation are possible consequences (i.e., harms) of a hazardous event involving electrical shock, and death is one possible outcome.

This report uses the term *risk control measure* for the actions taken to prevent or minimize the probability of a hazardous situation from occurring, to reduce its severity if it does occur, or to do both. This term can be used to cover what some engineers and regulators refer to as “mitigations” or “safeguards.” This term is chosen both for consistency with ANSI/AAMI/ISO 14971:2000 and because of its broader focus on anything that reduces either the probability or the severity of a hazard.

4 Perspective 1: Basic concepts of medical device software risk management

ANSI/AAMI/ISO 14971:2000, *Medical devices—Application of risk management to medical devices*, recognized worldwide by regulators, is widely acknowledged as the principal standard to use when performing medical device risk management. ANSI/AAMI SW68:2001, *Medical device software—Software life cycle processes*, also recognized by the U.S. Food and Drug Administration (FDA), makes a normative reference to ANSI/AAMI/ISO 14971:2000 requiring its use. The organization and content of these two standards provides the foundation for this TIR.

ANSI/AAMI/ISO 14971:2000 provides a framework and taxonomy for *medical device* risk management but does not provide details or explanations of its requirements in the context of software development. ANSI/AAMI SW68:2001 focuses on software development processes with specific references to risk management activities within each development process. However, ANSI/AAMI SW68:2001 does not provide detailed software risk management methods or fully explain how software risk management integrates into overall *medical device* risk management. This report attempts to address these details.

This section attempts to identify and explain key concepts relevant to reducing the risk of medical device software failures leading to unacceptable health risks.

4.1 Medical device risk management

The successful development of a safe and effective medical device requires the integration of a diverse set of knowledge and experience of personnel from a variety of disciplines such as clinicians, human factors engineers, biomedical engineers, electrical engineers, mechanical engineers, and software engineers. Experts from each domain will look at risk management with a different perspective, which is based on their particular experience. The goal of medical device risk management is to integrate each of these perspectives in such a manner that the device developed is as safe and effective as can be reasonably expected. Key to the success of this effort is a clear understanding, by all involved, of the device's intended use. With this understanding, the domain experts can begin to reason together about how each can best contribute to the overall safety and effectiveness of the device. One hopes it is self-evident that if these experts do not communicate effectively, the likelihood of the device performing as intended will be diminished.

By establishing the device's intended use, designers can begin to work on functional system requirements and their associated system boundaries, such as the device interface (human factors), hardware, software, and “networked” accessories. As these system boundaries begin to take shape, a basis is established for risk management processes. With this understanding, decisions can be made regarding how best to implement a risk control measure (e.g., hardware, software, information, or some combination of these).

4.1.1 Software risk cannot be managed effectively in isolation

Software can be a part of a medical device, an accessory to a medical device, or a medical device itself. Adequate medical device software risk management cannot be performed effectively by analyzing software alone, even in the case in which stand-alone software is considered a medical device. This type of device software runs on one or more computer and operating system platforms. The potential failure modes of these platforms, their configuration management, and the user environment must also be considered in order to perform adequate risk management.

On the one hand, software risk management cannot be performed in isolation from overall medical device risk management, while on the other hand, some special focus and knowledge are needed to adequately accomplish the software aspects of risk management.

Closer study of the purposes of software risk management activities reveals that the *system* is mentioned as often as the *software*. One cannot identify device *harms* by examining only software. Similarly, *hazards* may result from software events (causes) as well as hardware events, user events, or other nonsoftware events. Similarly, the protective measures—or risk control measures—may be designed into the software, or they may be designed into the hardware, documentation, or other nonsoftware element of the system.

Since this report focuses on software, it may seem as though software risk management can be a stand-alone process. To isolate software risk management from *medical device* (system) risk management would result in an incomplete device analysis, with a myopic view of the system, and a likelihood of lost safety requirements. The

software risk management activities are simply that part of system-level risk management that is related to software causes of—and software risk control measures for—medical device hazards.

4.1.2 Software input is an important part of device risk management

Although this report focuses on software risk management, there are many opportunities for software designers to contribute to the overall safety of the medical device during the early stages of device design. One should not wait to consider software until after key decisions about the design of medical devices are made. Doing so could result in missed opportunities for device-level risk control measures or inadequate understanding of risk during critical early requirement and design decisions.

By participating in the medical device design process, the software engineer can contribute to safety-related decisions concerning software risks and risk control measures as the design evolves. Additionally, the software engineer can encourage a safety-related decision involving the hardware–software partitioning and other aspects of the device design, functionality, and intended-use environment and computer platform.

Examples of software-related questions one could ask at the medical device design level include:

- Is a particular hazard best controlled using redundant hardware as opposed to software checks?
- What is the safe or safest state for this device?
- What are the special considerations for the power-up and power-down of this device? What is the hardware designer expecting the software to do during these times, and what power-up self tests are expected of the software?
- If nonvolatile storage is being used, what hardware interfaces are present, or should be present, to allow the software to properly manage nonvolatile storage when the device is powered down and powered up?

4.2 Software risk management

From a process perspective, software risk management for medical devices can be seen as the application of ANSI/AAMI/ISO 14971:2000 risk management requirements to the development of software. Software can be a part of a medical device, an accessory to a medical device, or a medical device itself.

One can assume that software cannot directly cause harm (it causes harm through hardware or provision of misleading information to users). Therefore, one should identify hazards (at the medical device level) and then look for potential software failures that could cause the hazards in isolation or through a chain of events or combination of conditions.

Where the software is itself a medical device intended to be distributed for use on commercial computer and operating systems, it is sometimes unclear where to draw the boundaries for software risk management. The fact that the hardware and operating system are not packaged with the software is no reason for the medical device designer to ignore hazards that could result from faulty platforms or poor administrative control and maintenance of these platforms by the users (e.g., security, virus checking, configuration management).

An important part of the software design and risk management process is that potential causes for hazards be considered in the intended-use environment. For instance, one may restrict the platforms to those that meet certain specifications included in the medical device labeling. Additionally, one might build checks into the software for acceptable platforms and for detection of platform failures.

While software aspects of risk management cannot be effectively performed in isolation from overall medical device risk management,³ there are activities that may be best performed by software engineers as an integral part of the software life cycle. There are also elements of software risk management⁴ that require more focus and different handling than that provided in ANSI/AAMI/ISO 14971:2000 for medical device risk management overall. We refer to this aspect of medical device risk management simply as *medical device software risk management*. It is important to stress that even the software aspects of risk management need to focus on the risk of harm—not just risk of software failure—in order to be effective.

³ Because of the interdependence of hardware failures, software failures, and hardware and software risk control measures.

⁴ For instance, all software failures are systematic, not random (as many types of hardware failures or breakdowns are), and their probability cannot be accurately estimated. Therefore, the way in which the probability component of risk is applied to software is quite different.

When one is discussing the potential for software to cause a hazardous situation, it is useful to develop a framework to support the discussion. Figure 2 illustrates such a framework in simple terms. First, we consider a system-level view of the software. From this view, we can see that hazards could be caused by software. Likewise, hazards could potentially be prevented, corrected, or mitigated by software.

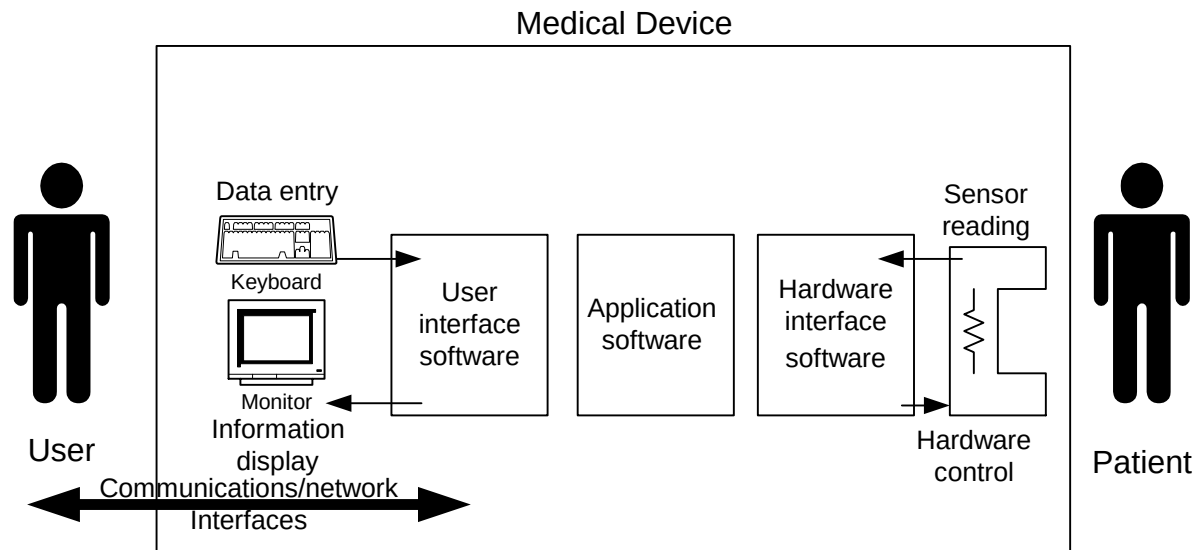


Figure 2—Software risk management context diagram

For example, in a patient-monitoring device that provides clinicians with vital signs data, the software responsible for displaying the vital signs data on the instrument could have a defect that causes the wrong value to be displayed. This defect creates a hazardous situation. If we assume the operator acts on this information, patient harm could occur either through improper treatment or delay in treatment. In this example, software is the cause of the hazardous situation. As another example, consider that the pressure in the cuff used for noninvasive blood pressure measurements could be monitored with a pressure transducer by software. If the software detects a pressure greater than a predetermined safety threshold, the software could take some protective action, thus mitigating the hazard of failed hardware that could result in uncontrolled pressurization of the cuff. In this example, the software is a protective measure for the system.

From a system-level view, we understand that software, in and of itself, cannot cause harm. However, that software, when controlling a hardware device in contact with the patient (depicted on the right side of Figure 2), or when reporting diagnostic information or treatment information (depicted on the left side of the figure), through a communication interface to another device or system (not shown), or through some environmental condition (not shown), could contribute to a hazardous situation if no other risk control measures are in place. Software may be used to implement risk control measures for hardware failures, so defects in this protective software can also affect safety.

Additionally, we need to identify software defects that could cause a hazard (or be in a causal chain that could result in a hazardous situation).

There are several distinctions in types of software functionality that are useful in software risk management:

- software intended to perform functions to implement clinical requirements for the intended use of the device (e.g., software calculating a radiation dosage);⁵
- software intended to implement a risk control measure; and
- other software.

Figure 3 depicts how these and several other distinctions should be used to identify software that is safety related.

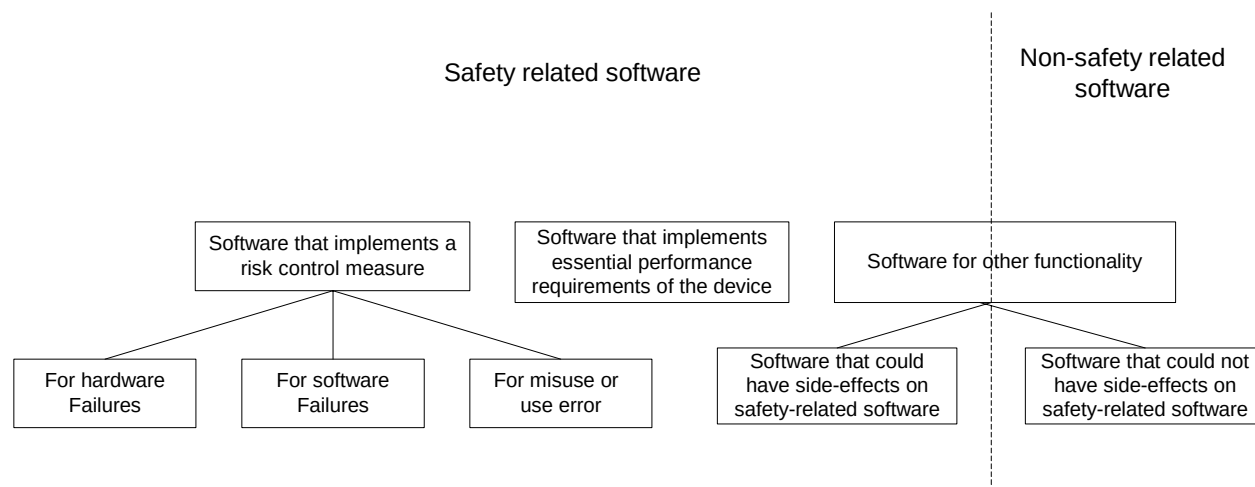


Figure 3—Types of functionality

Failure modes of all types of software components related to safety and effectiveness should be analyzed to determine appropriate risk control measures. If we look strictly from a functional perspective, it is easy to miss portions of the software that are safety related since some software is safety-related only from the perspective of side effects or defects that could inadvertently affect other safety-related software. This type of functionality is indicated on the right side of Figure 3.⁶

An identification of safety-related software should be done from both logical (i.e., functional) and physical (i.e., component) perspectives. From a functional or logical perspective, some software requirements may be directly related to safety and effectiveness, and others may not. From a physical or component perspective, some software components may be directly related to safety in terms of their purpose, others may be indirectly related in terms of potential predictable or unpredictable side effects, and still others may be unrelated and unable to affect safety if isolated by system design (e.g., running on a separate central processing unit (CPU) with a totally independent and protected memory and disk space). The results of this analysis allow identification and tracking of safety-related code or modules or components as well as detailed analysis of possible defects that could result in hazards.

From a physical perspective, it can be useful to assign risk or hazard severity ratings to software items or modules to distinguish highly critical components (e.g., those that could result in death or serious injury if they are defective) from components that could not affect safety. Doing so can serve as a basis for greater rigor and focus in verification, risk control, and configuration management activities for more critical components. If this is done, side effects need to be carefully considered, and the less critical components must be rated the same as any more critical components they could affect.

Off-the-shelf software used in a medical device could fall into one or more of the boxes in Figure 3. When it does, it must be handled accordingly without the presumption that such software is outside the scope of risk management or that risk control measures are not possible for such software.

⁵ This is sometimes referred to in IEC 60513:1994 as *essential performance*.

⁶ If there is no medical device-level isolation of software components, any defective component could affect other safety-related components and must itself be considered safety related.

We must note that software components could be identified initially as safety related. Subsequently, through certain risk control measures or design choices, the residual risk could be reduced to a level that the same software components could be treated as less critical or, in some cases, even as non-safety related (if the risk control measure entirely eliminates its potential hazard). Properly performed risk management results in reducing safety-related software to the smallest possible subset through isolation and inherently safe design.

A key aspect of software risk analysis is identifying potential causes of hazards. In this context, cause refers to software errors of commission or omission. This task may not be a simple one, because for each identified hazard there can be a single cause or multiple causes. Causes may exist singly or in multiple causal chains. These causal chains can be coupled or uncoupled or the result of a temporal condition. A simple causal chain example might consist of a buffer pointer overflow, which corrupts data in the next memory location, which results in a reasonable data value, but not for the current patient displayed on a screen, and which is acted on by the user. The concept of causal chains can be useful in determining effective and efficient risk control measures, as discussed in the next section and elsewhere in this report. Figure 4 illustrates this concept.⁷

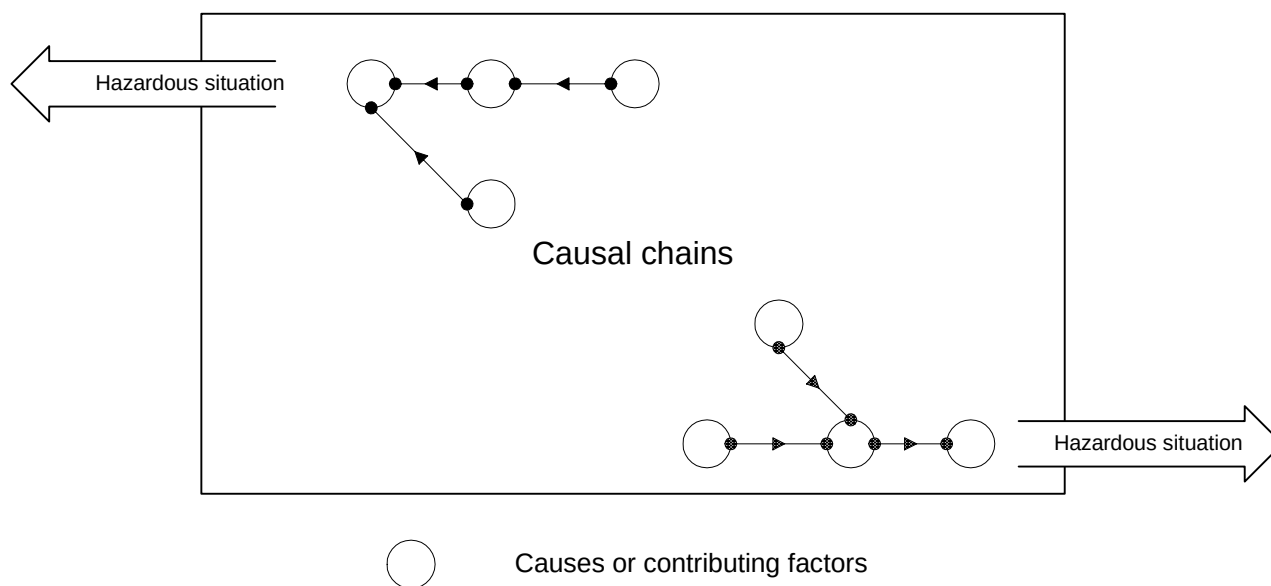


Figure 4—Causal chains

Direct versus indirect causes

When identifying potential software causes of hazards, one shall focus attention on software directly related to the clinical functionality of the device (e.g., an algorithm that calculates glucose levels in blood). Such software, if implemented incorrectly, can directly cause a device hazard. Examples of such “direct causes” include incorrectly implemented algorithms, incorrectly presented information, errors in information obtained from sensors, errors in information sent to actuators, and others. It is equally important to consider software causes that could indirectly result in device hazards. Examples of such “indirect causes” include uninitialized pointers, memory corruption, stack overflow, race conditions, and others. Although indirect causes for hazards may generally be unpredictable, appropriate attempts should be made to detect their occurrence and control any resulting hazards.

Annex A provides examples of specific causes based on categorization of typical medical device software functionality and relevant risk control measures that could be considered. Annex B provides examples of indirect causes for hazards attributable to unpredictable side effects and relevant risk control measures that could be considered. Neither of the tables in the annexes is exhaustive, and they should not be used as an exhaustive checklist. The tables are intended to trigger discussion and to serve both as an aid to identifying relevant causes and as a basis for challenging the thoroughness of the results of existing risk management processes.

⁷ The intersection at a common node could be an “AND” or “OR” condition, and as many possible sequences and necessary and sufficient conditions could apply.

4.3 Risk control

The purposes of medical device software risk management are to

- avoid, prevent, or mitigate the conditions under which software use, misuse, or defects could lead to a hazardous situation;
- reduce the severity of the hazard if such an event should occur; and
- minimize the probability of a hazardous situation caused by misuse or defects.

To accomplish this, software-specific aspects of risk management should address the following:⁸

- The analysis of the clinical or other possible intended use⁹ of the device to identify clinical hazards and related software functionality that provides clinical information or control commands related to safety and effectiveness.
- The identification of potential software causes (including causes that are part of a chain of events attributable to hardware failures or user errors) that could contribute to a hazardous situation. This analysis should include failures in software functionality directly related to safety and side effects from other software, hardware, or user errors.¹⁰
- The identification of software components that implement or could affect functionality related to safety or to the implementation of risk control measures—sometimes referred to as *safety-critical* or *safety-related* components.
- The identification of hardware, software, and labeling risk control measures to prevent software errors from resulting in a hazardous situation or to reduce the probability of such errors.
- An evaluation of residual risk after risk control measures are identified and determination of whether the risk is acceptable given the intended use and target population of the device.
- The verification of risk control measures and safety-related software components.

Information on the medical device evolves gradually as the risk management process is applied incrementally and iteratively over the development life cycle. The earlier in the development life cycle potential causes of hazards can be identified, the more likely inherently safe design choices can be made. Causes of hazards discovered late in the development life cycle are not likely to result in the most effective or efficient risk control measures. Section 6 of this report discusses risk management activities in relation to the software life cycle and provides examples of activities of risk management to consider at each stage of development.

Early in a project life cycle, from an analysis standpoint, it is often useful to consider the first and last points of potential risk control for each hazard. These will be referred to as the “first point of software control” (FPOSC) and “last point of software control” (LPOSC). In Figure 5, the first and last points of control are at the points of interface to the hardware on the left and right sides of the box.

⁸ Note that these areas are relevant both during initial design and development and during maintenance as each change is considered and implemented.

⁹ A device may be intended for a specific use for a specific target population, but if it could easily be used beyond this intended use, then relevant risks and risk control measures should be considered.

¹⁰ For instance, an uninitialized pointer in a module that results in overwriting of critical data or hardware memory failures that corrupt code or data.

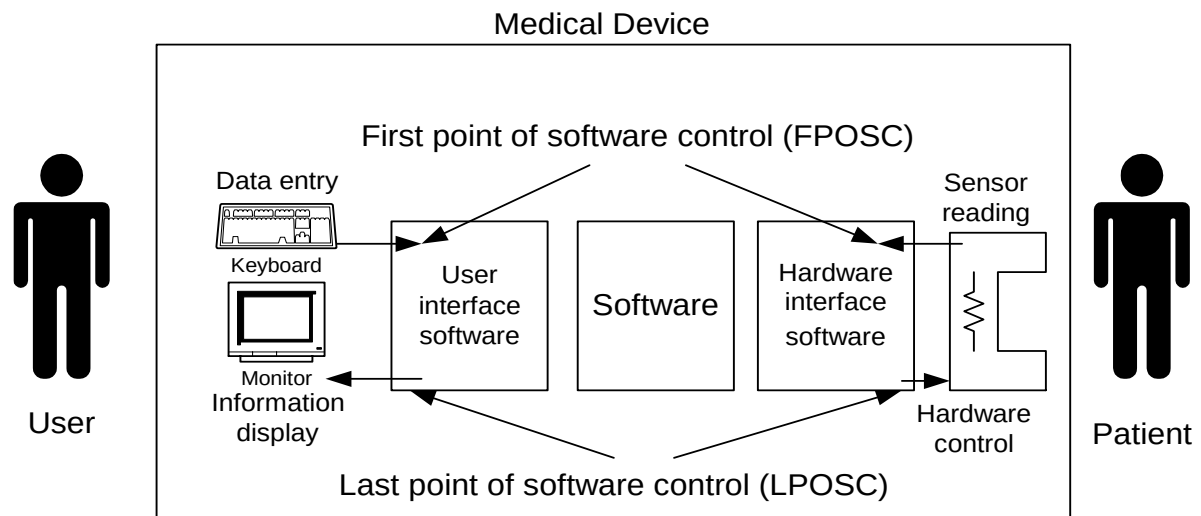


Figure 5—First and last points of control context diagram

A determination of the LPOSC versus the FPOSC in causal chains is highly subjective and purely pragmatic.¹¹ However, considering the device design in this context may provide insights that are useful for analyzing risk and identifying the best risk control measures.

A more detailed view of the concept of causal chains of events and first and last points of software control is provided in Figure 6. In this figure, each of the bubbles represents an error that is a potential cause. The chains indicate that multiple chains could create a hazardous situation. Generally, several complementary hazard identification methods are needed to help identify nonobvious causes and causal chains. If there is an adequate risk control measure at the end of the causal chain (LPOSC) that prevents the hazardous situation, then the individual links in the chain might not require separate risk control. For example, several links in the causal chain might represent erroneous states resulting from bad data elsewhere in the system. If data integrity can be ensured (FPOSC), then there may be no practical reason to put guard conditions around each possible intermediate error state.

¹¹ For instance, one might define an LPOSC for an infusion pump as the software that sends the commands to the pump motor, and an FPOSC might be at the user interface for entry of the infusion rate.