



BSI Standards Publication

Medical device software

Part 2: Validation of software for medical device quality systems

National foreword

This Published Document is the UK implementation of ISO/TR 8000-2:2017.

The UK participation in its preparation was entrusted to Technical Committee CH/62/1, Common aspects of Electrical Equipment used in Medical Practice.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

© The British Standards Institution 2017
Published by BSI Standards Limited 2017

ISBN 978 0 580 76425 7

ICS 35.240.80; 11.040.01

Compliance with a British Standard cannot confer immunity from legal obligations.

This Published Document was published under the authority of the Standards Policy and Strategy Committee on 31 October 2017.

Amendments/corrigenda issued since publication

Date	Text affected
------	---------------

Medical device software —

Part 2:

Validation of software for medical device quality systems

Logiciels de dispositifs médicaux —

*Partie 2: Validation des logiciels pour les systèmes de qualité des
dispositifs médicaux*



Reference number
ISO/TR 80002-2:2017(E)

AVA Documents
Document Sharing Hub
© ISO 2017



COPYRIGHT PROTECTED DOCUMENT

© ISO 2017, Published in Switzerland

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Ch. de Blandonnet 8 • CP 401
CH-1214 Vernier, Geneva, Switzerland
Tel. +41 22 749 01 11
Fax +41 22 749 09 47
copyright@iso.org
www.iso.org

Contents

Page

Foreword	iv
Introduction	v
1 Scope	1
2 Normative references	1
3 Terms and definitions	1
4 Software validation discussion	1
4.1 Definition	1
4.2 Confidence-building activities: Tools in the toolbox	1
4.3 Critical thinking	2
5 Software validation and critical thinking	2
5.1 Overview	2
5.2 Determine if the software is in scope	6
5.2.1 Document a high-level definition of the process and use of the software	6
5.2.2 Regulatory use assessment	6
5.2.3 Processes and software extraneous to medical device regulatory requirements	6
5.3 Development phase	7
5.3.1 Validation planning	7
5.3.2 Define	7
5.3.3 Implement, test and deploy	11
5.4 Maintain phase	13
5.4.1 Entering the maintenance phase	13
5.4.2 Planning for maintenance	14
5.4.3 Types of maintenance within the maintain phase	15
5.4.4 Process changes: Change to risk control measures	15
5.4.5 Emergency change	15
5.4.6 Maintaining for intended use	16
5.5 Retirement phase	16
6 Documentation	16
7 Prerequisite processes	17
Annex A (informative) Toolbox	18
Annex B (informative) Risk management and risk-based approach	24
Annex C (informative) Examples	28
Bibliography	84

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO should not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see the following URL: www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/TC 210, *Quality management and corresponding general aspects for medical devices*, in collaboration with Technical Committee IEC/TC 62, *Electrical equipment in medical practice*, Subcommittee SC 62A, *Common aspects of electrical equipment used in medical practice*, in accordance with ISO/IEC mode of cooperation 4.

A list of all parts in the ISO 80002 series can be found on the ISO website.

Introduction

This document has been developed to assist readers in determining appropriate activities for the validation of process software used in medical device quality systems using a risk-based approach that applies critical thinking.

This includes software used in the quality management system, software used in production and service provision, and software used for the monitoring and measurement of requirements, as required by ISO 13485:2016: 4.1.6, 7.5.6 and 7.6.

This document is the result of an effort to bring together experience from medical device industry personnel who deal with performing this type of software validation and who are tasked with establishing auditable documentation. The document has been developed with certain questions and problems in mind that we all go through when faced with validating process software used in medical device quality systems such as the following: What has to be done? How much is enough? How is risk analysis involved? After much discussion, it has been concluded that in every case, a set of activities (i.e. the tools from a toolbox) was identified to provide a level of confidence in the ability of the software to perform according to its intended use. However, the list of activities varied depending on factors including, among others, the complexity of the software, the risk of harm involved and the pedigree (e.g. quality, stability) of vendor-supplied software.

The intention of this document is to help stakeholders, including manufacturers, auditors and regulators, to understand and apply the requirement for validation of software included in ISO 13485:2016, 4.1.6, 7.5.6 and 7.6.

Medical device software —

Part 2:

Validation of software for medical device quality systems

1 Scope

This document applies to any software used in device design, testing, component acceptance, manufacturing, labelling, packaging, distribution and complaint handling or to automate any other aspect of a medical device quality system as described in ISO 13485.

This document applies to

- software used in the quality management system,
- software used in production and service provision, and
- software used for the monitoring and measurement of requirements.

It does not apply to

- software used as a component, part or accessory of a medical device, or
- software that is itself a medical device.

2 Normative references

There are no normative references in this document.

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO 9000 and ISO 13485 apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

- IEC Electropedia: available at <http://www.electropedia.org/>
- ISO Online browsing platform: available at <http://www.iso.org/obp>

4 Software validation discussion

4.1 Definition

The term “software validation” has been interpreted both broadly and narrowly, from just testing to extensive activities including testing. This document uses the term software validation to denote all of the activities that establish a level of confidence that the software is appropriate for its intended use and that it is trustworthy and reliable. The chosen activities, whatever they might be, should ensure that the software meets its requirements and intended purpose.

4.2 Confidence-building activities: Tools in the toolbox

The tools in the toolbox (see [Table A.1](#) to [Table A.5](#)) include activities completed during the life cycle of software that reduce risk and build confidence.

4.3 Critical thinking

This document promotes the use of critical thinking to determine which activities should be performed to adequately validate specific software. Critical thinking is a process of analysing and evaluating various aspects of software, as well as the environment in which it will be used, to identify the most meaningful set of confidence-building activities to be applied during validation. Critical thinking avoids an approach that applies a one-size-fits-all validation solution without thoroughly evaluating the solution to determine if it indeed results in the desired outcome. Critical thinking recognizes that validation solutions can vary greatly from software to software and also allows for different validation solutions to be applied to the same software in a similar situation. Critical thinking challenges proposed validation solutions, to ensure that they meet the intent of the quality management system requirements, and considers all key stakeholders and their needs. Critical thinking is also used to re-evaluate the validation solution when characteristics of the software change, when the software's intended use changes or when new information becomes available.

Critical thinking results in a validation solution that establishes compliance for a manufacturer, ensures that the software is safe for use, results in documented evidence that is deemed appropriate and adequate by reviewers, and results in a scenario in which individuals performing the validation work feels that the effort adds value and represents the most efficient way to reach the desired results.

[Annex C](#) presents example studies demonstrating how critical thinking can be applied to software validation of software used in medical device quality systems in a variety of situations, including different complexities, pedigrees and risk levels.

5 Software validation and critical thinking

5.1 Overview

Throughout the life cycle of software for medical device quality systems, appropriate controls need to be in place to ensure that the software performs as intended. Incorporation of critical thinking and application of selected confidence-building activities result in establishing and maintaining a validated state of the software. [Figure 1](#) depicts a conceptual view of typical activities and controls that are part of the life cycle from the moment the decision is made to automate a process until the software is retired or is no longer used for medical device quality systems. Although [Figure 1](#) depicts a sequential model, in reality, the process is of an iterative nature as elements are defined, risks are identified and critical thinking is applied.

When developing software for use in the medical device quality system, a fundamental confidence-building activity to be selected from the toolbox is the choice of software development life-cycle model. The model chosen should include critical thinking activities that enable the selection of other appropriate tools during various life-cycle activities. The results of the analyses and evaluations used drive the selection of the most meaningful set of confidence-building activities to ensure that the software performs as intended. This document does not mean to imply or prescribe the use of any particular software development model. For simplicity, however, the remainder of this document explains the concepts of critical thinking within the context of a waterfall development model using generic names for the phases. Other software development models (e.g. iterative, spiral) can certainly be used as long as critical thinking and the application of appropriate tools are incorporated into the model.

Software for medical device quality systems

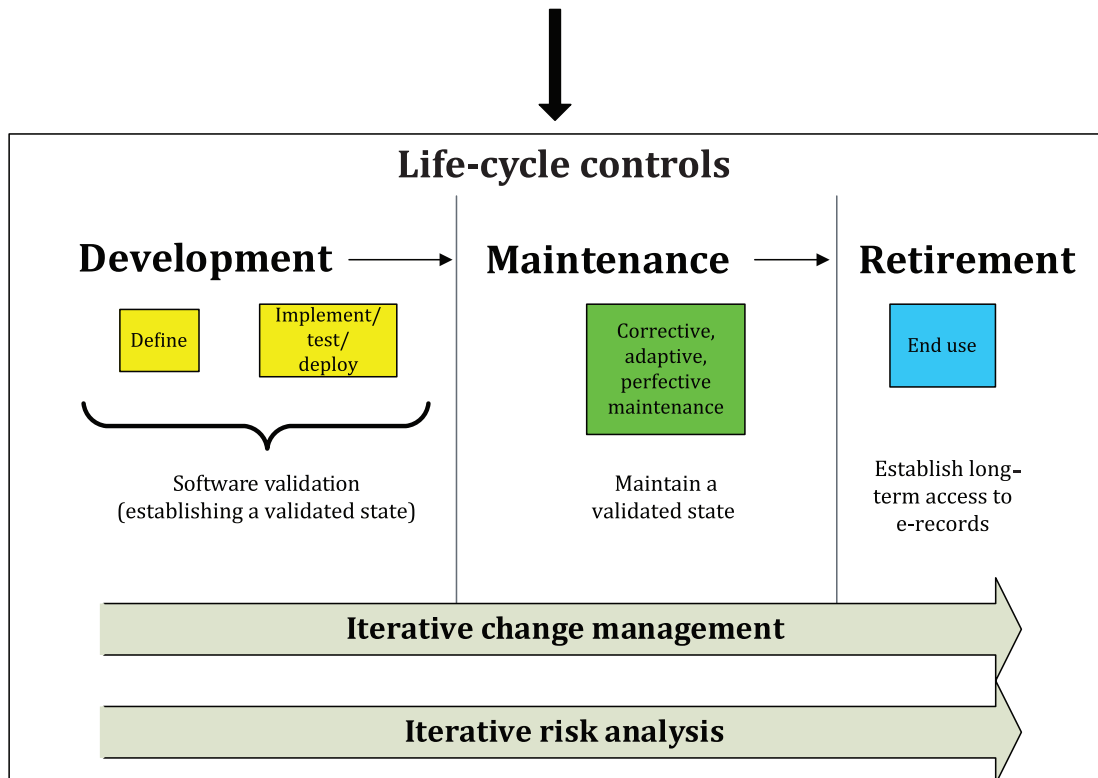


Figure 1 — Life-cycle controls

When considering using software in a process, one should identify whether the proposed software is used as part of a medical device quality system process through an investigation of its intended use. If so, then the software should be validated for its intended use. Although this document describes an approach to validating software for medical device quality systems, the same approach is also good practice for software to evaluate whether it fulfils defined requirements. The most critical part of software validation is developing/purchasing the right software tool to be able to support processes as intended by the manufacturer. This implies that requirements should be determined accurately to evaluate whether the developed/purchased software is suitable to fulfil the requirements of the intended use. Technical requirements suitable for verification, as well as process requirements suitable for validation, are equally important. When considering using software in a process, the software can interact or can have interfaces with other software.

During the development phase of the life cycle, risk management and validation planning tasks are performed to gather information and drive decisions in the following four areas:

- level of effort applied and scrutiny of documentation and deliverables;
- extent of content in the documentation and deliverables;
- selection of tools from the toolbox and methods for applying the tools;
- level of effort in applying the tools.

The primary drivers for decisions in the four areas are process risk and software risk. However, other drivers can influence decisions, including the complexity of the software and process, the type of software and the software pedigree.

The validation planning process consists of two distinct elements. The first validation planning element involves determining the level of rigor in the documentation and the scrutiny to be applied to the review of the resulting deliverables. The decisions in this element are primarily driven by the results

of the process risk analysis. The second validation planning element drives the selection of tools from the toolbox to implement, test and deploy the software. The choice of tools is driven primarily by the software risk analysis. Such planning steps result from different types of risk analyses and are depicted as separate activities in this document. However, many times the steps are combined into one activity, which includes the different aspects of risk analysis and the resultant choices for proceeding with validation.

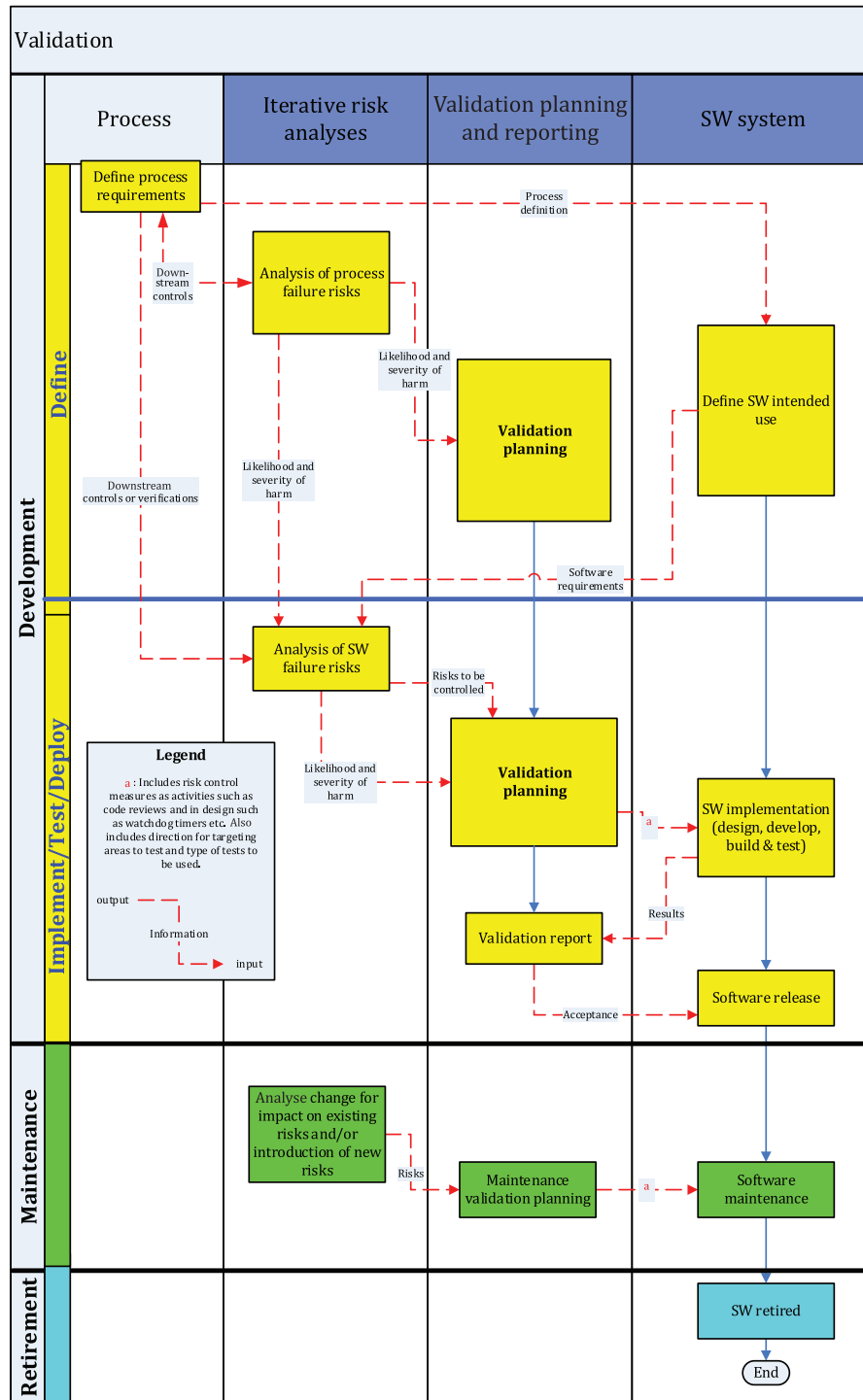
During the development phase of the life cycle, risk management and validation planning tasks are used to define the appropriate level of effort to be applied to the software and to determine what confidence-building tools to apply. This type of approach results in the completion of appropriate value-added activities and verification tasks, which are the basis for establishing a validated state. Once these activities and tasks are executed, the tools and their associated results are cited in a validation report as support for the conclusion that the software is validated.

Once deployed, the software moves into the maintenance phase of the software life cycle. During this period, the software is monitored, enhanced and updated as dictated by the business needs or regulatory requirement changes. Change control activities use the same concepts as the initial approach that was applied during the development phase of the life cycle. Changes, however, are now assessed as to their effect on the intended use, on the risk of failure, on the risk control measures that were applied during the initial development and on any functionality of the software itself.

The retirement phase is the act of removing software from use either by removal of the process or by replacement of the software being used for the process.

The activities shown in [Figure 1](#) reflect the primary software life-cycle control activities. Other work streams include project management, process development, vendor management (if applicable), and possibly others, depending on the software being implemented.

[Figure 2](#) depicts software life-cycle control activities and critical thinking within the context of other work stream activities. The critical thinking activities appear in the iterative risk analysis and validation work streams. It is important to have clear and formal definitions of these work streams within the organization's business model to ensure that a program properly manages the software from both business and regulatory perspectives.



NOTE When the term "develop" or "development" is used, it is about the development of a validated state of the software.

Figure 2 — Life-cycle controls work stream

The various colours depicted in Figure 2 correspond to the life-cycle portion that is shown in the overall approach flow chart in Figure 1. The red dashed lines indicate information that is outputted from one activity and that provides input to or helps drive decisions in another activity. The diagram demonstrates how the ordering of the activities is driven by the need to have input information before completing the activities that require the input. It is important to note that all the activities are completed irrespective of the size or complexity of the software being implemented. However, for

larger or more complex software, such activities will most likely be discrete; for smaller or simpler software, many of those activities will be combined or completed simultaneously.

In summary, the critical thinking approach described a systematic method for identifying and including appropriate confidence-building activities or tools in various work streams to support the conclusions that the software is validated on release and that the validated state will be maintained until the software is retired.

The following subclauses provide additional details for each of the blocks found in the life-cycle controls depicted in [Figure 1](#). The subclauses use the work stream depiction of iterative risk analyses, validation and software activities shown in [Figure 2](#) to provide perspective on the various decision points and decision drivers that incorporate critical thinking.

5.2 Determine if the software is in scope

5.2.1 Document a high-level definition of the process and use of the software

The first step in determining whether the software is considered to be used for medical device quality systems is to document a high-level definition of the process and use of the software. This activity might seem of small value when it is readily known that the software is in scope and one is already embarking on defining the full intended use of the software. However, for situations in which such assumptions are less clear, documenting the process and use enables the clear determination as to whether the software is in scope. In addition, for identified out-of-scope software, such an activity can result in a rationale as to why the software is out of scope.

5.2.2 Regulatory use assessment

A regulatory use assessment can be used to determine whether the software is a “software for medical device quality system” and therefore falls within the scope of this document. Start by identifying the specific regulatory requirements that apply to the processes that use the software and the data records that are managed by the software. A series of questions can be used to help fully understand the role that the software plays in support of these regulations. The following types of questions should be considered.

- a) Could the failure or latent flaws of the software affect the safety or quality of medical devices?
- b) Does the software automate or execute an activity required by regulatory requirements (in particular, the requirements for medical device quality management systems)? Examples may include capturing electronic signatures and/or records, maintaining product traceability, performing and capturing test results, maintaining data logs such as CAPA, non-conformances, complaints, calibrations, etc.

A “yes” answer to any of the questions identifies software that is required to be validated and is within scope of this document.

At times it can be difficult to determine whether a process and corresponding software are part of the quality system. Some tools can have many degrees of separation from the actual medical device. Each organization should, therefore, carefully consider the circumstances surrounding such borderline software and should completely understand the impact of the failure of the software on the processes and, ultimately, on the safety and efficacy of any manufactured medical devices. When the answer is not certain, the best approach is to consider the software as in scope and to apply the approach defined in this document.

5.2.3 Processes and software extraneous to medical device regulatory requirements

When processes or software contain functionality that falls outside of medical device regulatory requirements, an analysis should be performed to determine which parts of the software are considered to be in scope and which parts are not in scope. Such decisions should be rationalized on the basis of the degree of integration between various components, modules and data structures of the software and in

accordance with the compliance needs of the organization. This rationalization is especially important in the case of software used in support of the quality system, such as large, complex enterprise resource planning (ERP) software. ERP software can include functionality for non-medical device-regulated processes such as accounting and finance. Although such functionality can be crucial for business operations and have to meet certain government requirements (e.g. those of the Sarbanes-Oxley Act).

5.3 Development phase

5.3.1 Validation planning

The first part of the validation planning activity captured when critical thinking is applied, uses input from the process risk analysis (see [Annex B](#)) to establish the basis for the level of effort that should be applied to the documentation and to drive the choices of tools from the Define section of the toolbox (see [Table A.1](#) to [Table A.5](#)). The second part uses input from the software risk analysis to drive the choices of the implement, test and deploy tools from the toolbox. Once executed, the activities and the validated state of the software are established, and evidence of the validation is documented in the validation report.

Many development life-cycle models can be applied during the development phase. None is advocated or recommended by this document; however, application of a controlled methodology is expected. Such a controlled methodology would be based on the concept of defining requirements (including intended use), before implementation, testing and deployment, which are fundamental to establishing the validation of the software for its intended use.

5.3.2 Define

5.3.2.1 Define block requirement

The activities completed within the define block include the definition of the process, the definition of the software intended use within that process and the planning for the level of validation effort based on the inherent risks identified within the process. [Figure 3](#) depicts this portion of the development phase within the selected waterfall model example.

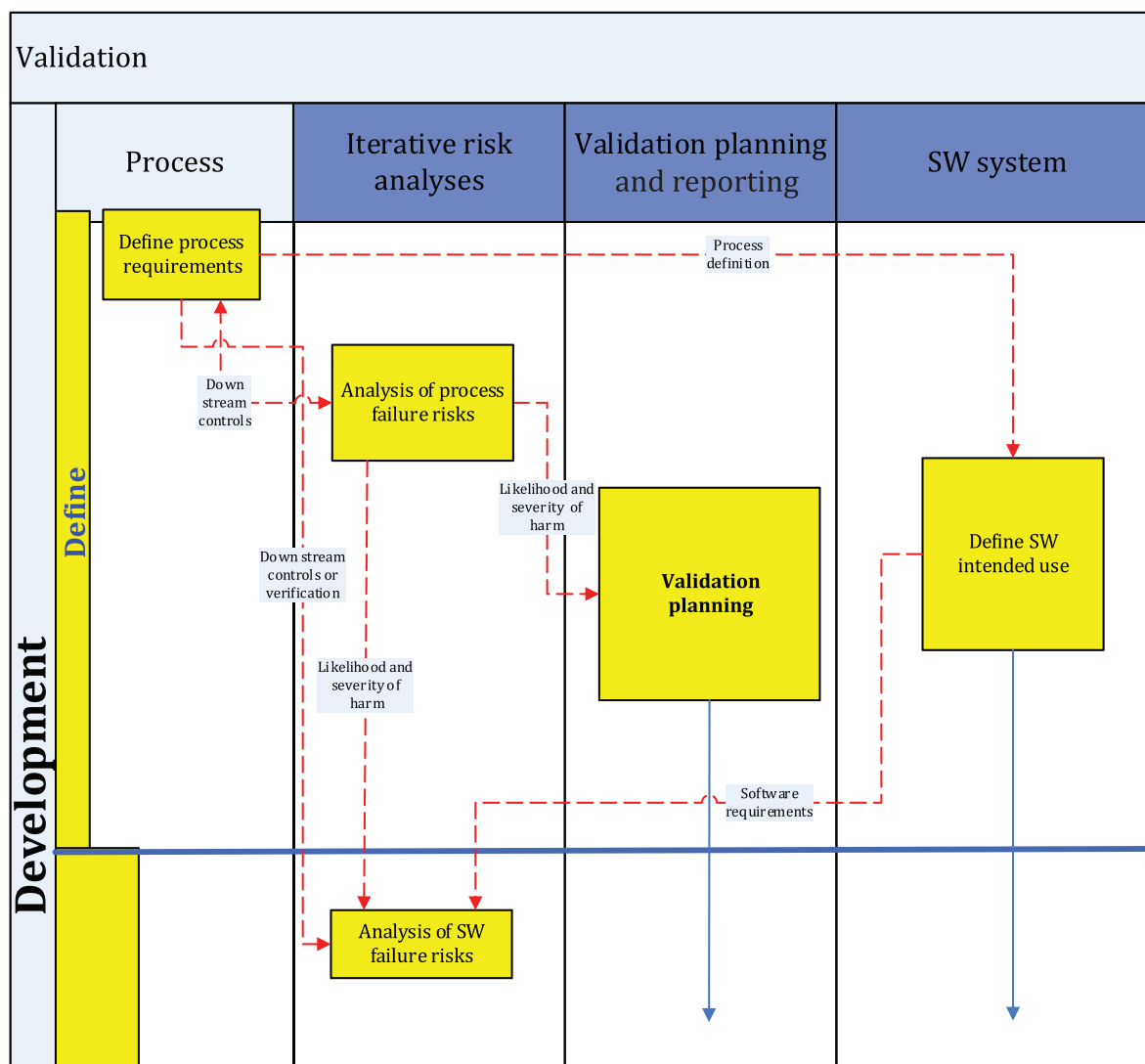


Figure 3 — Life-cycle phase: Define block work streams

5.3.2.2 Process requirements

The first step in the application of life-cycle controls is to define the purpose and function of the entire process, particularly the portions intended to be controlled by the software. This is best performed by involving the appropriate subject matter experts and including all aspects and activities associated regardless of whether all will be controlled by the software. Benefits are explained below:

- regulatory requirements can be clearly discerned;
- intended use of the particular software within the context of the process can be clearly discerned;
- process aspects and activities not controlled by the particular software can be clearly identified and addressed procedurally or by some other means;
- process activities upstream and downstream from the software are identified and can be considered when assessing the risks of the software failure and in devising risk controls for software failure.

The process definition activity establishes the foundation for decisions that are made later in the life cycle and is essential to targeting efforts on value-added, risk-based activities.

5.3.2.3 Analysis of process failure risk

The relationship of the software to the final safety and efficacy of the medical product will be considered during the risk analysis process. The following should also be considered.

- Risk of harm to humans: This includes direct harm to patients and users, and indirect harm when software controlling manufacture or quality of the device malfunctions, resulting in failure of the device, which causes harm.
- Regulatory risk: Risk of non-compliance with regulatory requirements to be considered if failure of the software can lead to loss of records (e.g. CAPA, complaint, device master record or device history file records) required by regulatory agencies or to deviations from quality system and manufacturing procedures.
- Environmental risk: Risk to the environment in which the software operates. Both the physical and the virtual.

Other types of risks can be incorporated into this model. However, the scope of this document and the tools discussed to reduce risk do not address them. This document focuses on the determination of the human safety risks, regulatory risks and environmental risks associated with software failure within the context of process failure.

The results of risk analysis should be clearly documented because such results are valuable decision drivers for selecting tools from the toolbox and for justifying the level of effort applied to the validation activities.

5.3.2.4 Validation planning

The extent of confirmation and objective evidence needed to ensure that the requirements of the software can be consistently fulfilled depends on the critical value of the software within the overall process. Therefore, the first validation planning activity regarding the level of effort applied and the scrutiny of the deliverable elements is based solely on input from the process failure risk analysis.

This validation planning activity results in a first iteration of validation planning documentation. The planning includes the selections for “level of effort” (i.e. the decisions) and the rationale for those choices (i.e. the decision drivers). The rationale should be based on the risk of harm posed by a failure of the process. The validation plan should provide objective evidence of the application of critical thinking to the validation planning process.

5.3.2.5 Software intended use

5.3.2.5.1 Elements of intended use

The intended use is meant to provide a complete picture of the software functionality and its purpose within the process. Specifically, it is meant to describe and explain how the software fits into the overall process that it is automating, what the software does, what one can expect of the software and how much one can rely on the software to design, produce and maintain safe medical devices. The intended use is a key tool used to understand what potential risks are associated with the use of the software.

The three main elements of intended use are:

- purpose and intent related to
 - the software’s use (e.g. who, what, when, why, where and how),
 - the regulatory use of the software, and
 - the boundaries of the software within the process or with other software and/or users;
- software use requirements. As the complexity and, generally, the risk increases, this element adds more detailed information regarding the use of the software (e.g. use cases, user requirements).

- software requirements. As the complexity and risk increase to the point where clear direction should be provided to implementers of the software, this element provides more specific and detailed information regarding the expectations of the software.

The intended use should be formally controlled and approved by properly skilled and experienced personnel on regulations, quality system and the process being controlled.

Given that we should validate for “intended use”, validation cannot be accomplished unless the intended use for the software is sufficiently defined.

The following subclauses provide further details about the elements of the software intended use.

5.3.2.5.2 Software purpose and intent

It contains information covering three elements: software use, regulatory use and boundary definitions.

a) Software use

- When defining the use of the software, one should consider the following questions: what, why, how, who, where and when. The answers explore how the software is being used to meet the process requirements. Such exploration helps to identify base information for the definition of the software as shown in [Table 1](#).
- The answers that are meaningful to the description of the software should be included in the established intended use definition.

Table 1 — Sample questions and answers

Question	Answer
What problem is the software addressing?	There is a problem in efficiently and accurately pooling product defect data for trending purposes.
Why is the software useful?	The software enables the pooling and trending of data from global locations.
How does the software solve the problem?	The software drives the process of data collection and automatically pools and calculates trending information or the software does not drive the process but provides a passive collection of data used to pool and calculate trending information.
Who uses the software?	The Quality Assurance and Operations Departments use the software.
Where is the software used?	The software is accessed by locations in the United States, Europe and Japan.
When is the software used?	The software is accessed during normal business hours for the global locations (i.e. daily, Monday to Friday).
NOTE These sample questions are not exhaustive.	

b) Regulatory use

- When evaluating the regulatory use, one can further explore the questions answered to determine if the software is in scope (see [5.2](#)). Expand all “yes” answers to include the reasons for these conclusions. Now that the software has been identified as in scope, any potential harm to humans (other than the users of the medical device) or to the environment needs to be determined. All the following questions direct the user’s consideration towards elements that are required as part of regulations, such as public health, safety and validity or authenticity of electronic records and signatures.
 - How could the failure or latent flaws of the software affect the safety or quality of medical devices?
 - How does the software automate or execute an activity required by regulation, in particular, the requirements for medical device quality management systems?

- How could the software cause harm to people (other than the users of the medical device) or to the environment?
- c) Software boundaries
- Defining the parts of the process that are to be controlled through software (boundaries within the process) and the places where software interfaces exist (boundaries with other software) facilitates the effectiveness and efficiency of the validation efforts. For example, it often can be more efficient to validate multiple software products as a group rather than to perform individual validations. One should also consider how various grouping strategies can affect efficiencies of ongoing maintain phase activities.
 - Boundaries within the process
 - Identifying the boundaries of the software within the process clearly establishes the aspects that are to be included in the intended use. Software can automate an entire process or can automate a subset of activities and can also function as a repository of data for the process. Understanding the role that the software plays with in the process helps to determine the risks associated with a potential failure of the software.
 - Boundaries with other software
 - When externally interfaced with other medical device quality systems software or with medical device software, it is important to identify all interfaces between the applications. Validation efforts typically include the internal interfaces as an inherent part of the method but the software's external interfaces should not be ignored. All interfaces between software applications should be incorporated into the critical thinking process.

5.3.2.5.3 Software use requirements

Software use requirements consist of well-documented and traceable elements that provide an additional layer of details to the software's purpose and intent. Such requirements provide insight into the use scenarios of the system, from either a user's perspective or a product-needs perspective. The user's perspective can be captured in the form of user requirements, use cases or other user-centric definition of needs. The product-needs perspective captures the needs of the medical device that is being affected by the system and can, in some cases, include a reference to specific device requirements or a synopsis of the product lines that the software might affect.

5.3.2.5.4 Software requirements

Consists of activities to define elements, well-documented and traceable, that specify what the software needs to do in order to meet its purpose, intent and use requirements. Software requirements comprise the input to the system's design, to the system's configuration, as well as the input to testing activities.

5.3.3 Implement, test and deploy

5.3.3.1 Required activities

The activities completed within the implement, test and deploy block include

- a) planning of the level of validation rigor in the design,
- b) development and configuration,
- c) building of the software, and
- d) testing of the software based on risks identified.

5.3.3.2 Analysis of software failure risks

The key point of software failure risk analysis is to determine and document the inherent risks associated with software failure and to identify any control measures (including process and software controls outside the software under analysis). The analysis is then used to arrive at a realistic and effective validation approach.

When reviewing risks attributable to software failure, one considers any process controls outside the software under analysis that constitute risk control measures. Such risk control measures can reduce the impact of a software failure, thus reducing the dependency on the software and thereby reducing the reliance on testing (examination) and documentation (collection of objective evidence) to ensure the safe operation of the software. Including such considerations will help ensure that the software is viewed within the context of the overall process.

A model presented in [Annex B](#) does not represent an all-encompassing formula. The resulting analysis provides input into the choice of tools from the toolbox to be used for the software validation.

5.3.3.3 Validation planning

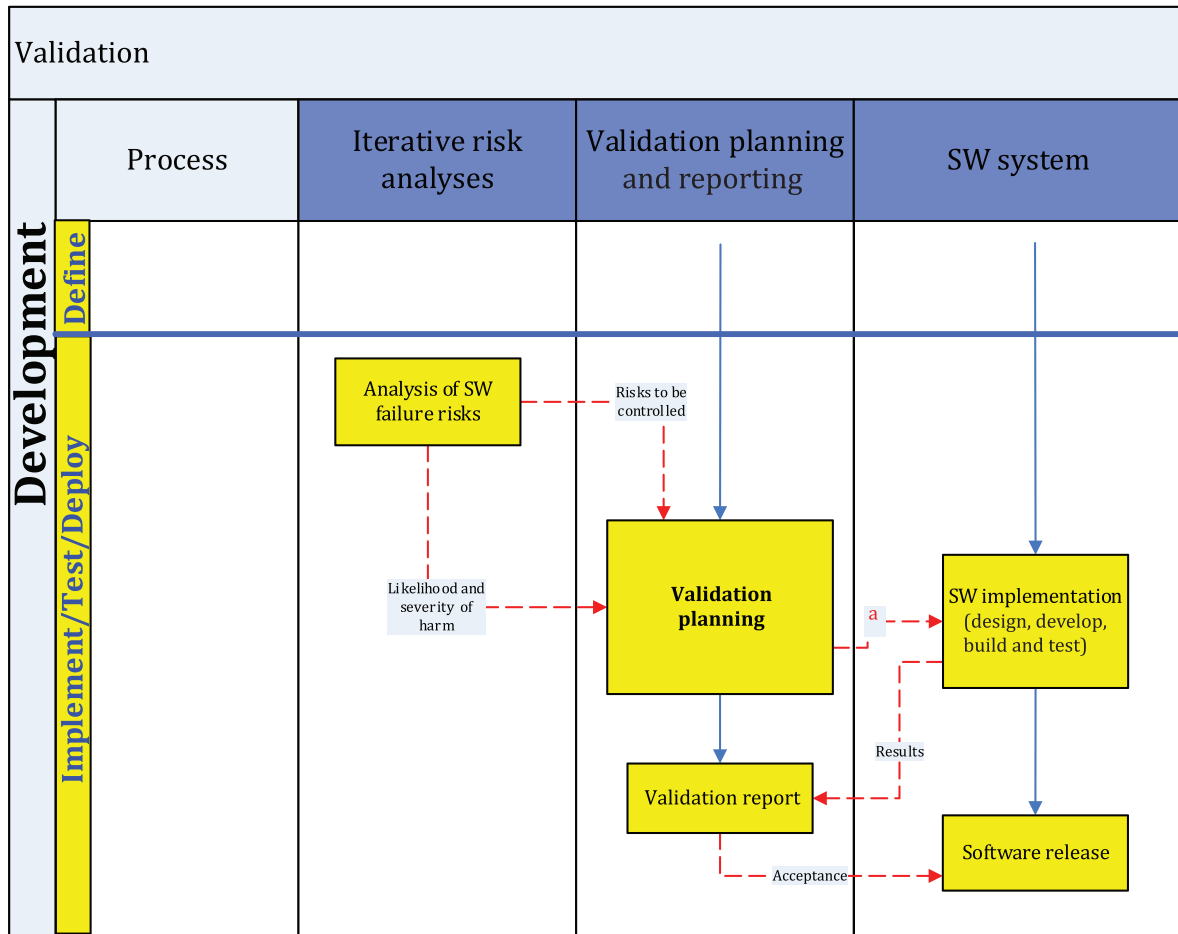
This activity uses the intended use definition and the results of the software risk analysis as inputs to the identification of risk control measures and the selection of tools from the toolbox that will be used to validate the software.

It is important that the tool selection process include qualified individuals who have an understanding of the impact of failure on the process and the inherent risks of failure of the software that will automate that process, although the individuals need not be software experts. Individuals from various disciplines (regulatory, quality, clinical and the like) should be involved in the planning process for any software that is highly complex or that has a high risk associated with its failure.

The validation planning activity results in a documented plan that describes the choices made (decisions) and the reasons for the choices being made (decision drivers). Validation planning provides documented evidence of the rationale used to select value-added confidence-building activities used to ensure that the software will perform as intended.

5.3.3.4 Software implementation (design, development, building and test)

This block includes the actual application of many tools from the toolbox. The tools (activities required as identified in the validation plan) are carried out during the design, development, building and test steps (see [Figure 4](#)).



- a Includes risk control measures as activities such as code reviews and in design such as watchdog timers, etc. Also includes direction for targeting areas to test and type of tests to be used.

Figure 4 — Life-cycle phase: Implement, test and deploy block work streams

5.3.3.5 Validation report

Once sufficient confidence-building activities, including tools selected from the toolbox, have been completed to ensure that the software performs as intended, the activities and (possibly) the results of the activities should be cited in a final validation report. The formal review and approval of the report provide a summary of references to all documented objective evidence that supports the conclusion that the software has been validated for its intended use.

5.3.3.6 Software release

Concluded the validation, a formal controlled method should exist for releasing the software. The defined controls should ensure and confirm that the software placed into use matches the software that has been assessed through the confidence-building activities cited in the validation report. If not, the rationales and controls should ensure and confirm that the results sufficiently represent the performance of the released software in its intended environment.

5.4 Maintain phase

5.4.1 Entering the maintenance phase

Phase entrance criterion: Software maintain phase starts after software is released for use.

Activities scope: Maintain phase activities consist of ensuring that the software remains in a validated state while accommodating, managing and controlling various types of changes. Some types can involve only changes to the process within which the software is used.

Changes to any validated system should be done in a controlled manner in accordance with policies and procedures.

Ideally, it is recommended that changes be made in a test environment and are validated before promoting the system to production use. When validating the change in a test environment is not possible and testing of changes should occur in the production environment, appropriate controls should be in place to minimize unwanted impacts to the production environment or directly to the product.

The selection of which tools from the toolbox are to be used in validating the change should be determined by the impact analyses of software changes on existing risk control measures by the introduction of new risks or by both.

Since the actual use of the software or its configuration can migrate over time despite efforts to control it, the use of maintain phase-specific tools, such as periodic monitoring of actual use or real-time monitoring of software configuration, might be appropriate. If a change in intended use results in a higher risk level, the change can trigger a more extensive set of validation activities than was originally performed, even without changes to the software.

Decisions regarding the choices and evidence of the execution of more extensive validation activities should be documented as part of validation planning to provide evidence that the software remains in a validated state.

5.4.2 Planning for maintenance

Maintenance planning evidence should be recorded before starting maintain phase.

Ideally, maintenance planning begins during the development phase. One should properly understand how changes will affect software validation, examine the effect of changes on risk and plan the proper activities to maintain validation.

Large and complex software might have to accommodate day-to-day maintenance and performance-tuning activities without affecting the software's ability to perform as intended. Planning for maintenance during the development phase can define which of the operational activities can be done without affecting validation and which changes require validation efforts. Before the software arrives at the maintain phase, methods of determining when to perform further validation activities on the software should be planned and discussed, including how changes in an underlying software (e.g. operating system, database management system) might affect the validated software. It is helpful to train software operators to recognize such boundaries and to recognize the difference between normal operational activities and any changes requiring validation.

Traceability analysis is a useful tool in managing maintenance activities. Traceability analysis is frequently a cornerstone of the initial validation and is often facilitated through a traceability matrix. The matrix maps requirements for tests or other verification activities, risk control measures and so forth. If performed well during the initial implementation, traceability analysis becomes a valuable tool during maintenance by facilitating the identification of the impact of changes and of the appropriate activities to validate the changes. In simple software, such analysis can consist of a single-level trace of requirements to implementation and verification. However, complex software can require a multilevel matrix that decomposes top-level functionality into lower-level requirements and then into implementation and verification. Other information can also be embedded, for example, sections of the software that are considered particularly high risk can be designated within the trace matrix, possibly with additional validation activities indicated.

5.4.3 Types of maintenance within the maintain phase

There are a number of reasons why the software would change after it is released for use. Some of the more common types of maintenance changes include the following:

- corrective maintenance changes made to correct errors and faults in the software;
- perfective maintenance changes to enhance performance, maintainability or other software attributes;
- adaptive maintenance done to update the software operational environment (e.g. changes to operating system, system hardware or other applications with which the software interfaces).

5.4.4 Process changes: Change to risk control measures

When the process that is wholly or partially controlled by software change, an impact analysis should be done to re-evaluate risk control measures.

The process that is wholly or partially controlled by software can change independently of the software. When a process change occurs, it is important to understand how that change affects the validated state of the software. The process change can affect the intended use of the software or of other supporting information regarding the software.

The process change can also affect the risk control measures in place for the software that are part of the validation rationale. Because the software is part of a process, downstream controls might be important risk control measures for the software. If the downstream controls are properly identified as part of the software validation rationale and process definition, the impact analysis for the proposed process change will be easier to perform. Impact analysis is essential to performing maintenance in a way that builds confidence in both the software and the process within which the software is operating.

5.4.5 Emergency change

Emergency changes should be governed by approved processes. These processes should require the justification for development and implementation, the mechanisms for gaining and recording authorization to deploy the change, the provisions to ensure that the risk has been properly assessed and controlled, and any activities necessary to invoke the emergency change (e.g. training, communication, product review and disposition). In this circumstance, performing the provisions for properly assessing and controlling risks represents the minimum set of activities needed to meet the regulatory requirement for the validation of changes prior to release.

Software changes might need to be performed under emergency circumstances. Typically, such changes are required if the integrity of the software, operating system or data has been compromised or to facilitate mitigation of potential harmful situations.

Additional, post-emergency change activities might be needed to fully evaluate all effects of the change. Depending on the overall risk posed by a failure of the process, process output (data or product) might need additional controls until all post-emergency change activities are complete.

Software problems that interrupt a process are usually obvious. Detecting subtle, underlying problems can be more difficult. Periodic evaluation of error logs, help centre requests, customer feedback and other defect reports can point to underlying problems. Such monitoring techniques can pick up problems that are not obvious enough to result in an error report but that can indicate correctable software problems. Maintenance activities might then be necessary to deal with identified problems by implementing corrections in future releases. Additionally, issues in released software that are attributed to these types of software problems can be proactively managed.

After the maintenance activities correct the problems for future releases, the historical impact of identified defects in released software should be reviewed and their consequences managed.

If the software validation depends on ensuring the correct usage of the software through training, periodic evaluation of user training effectiveness is another monitoring technique that helps to maintain the validated state.

5.4.6 Maintaining for intended use

If there is a change in the intended use of the software, it should be validated for the new intended use or the new use should cease. In the latter case, a risk assessment is in order to make sure no risks were introduced during the period of unauthorized use.

Change in intended use is a category that requires special attention because the change could be subtle and hard to detect or it could be quite obvious. In the subtle case, a change occurs to the purpose and intent or to the software use requirements and does not necessarily cause a change to the detailed software requirement element (see [5.3.2.5](#)). Such a change might occur intentionally or as a result of simply using existing software in a new mode without realizing that the intended use has been affected. Intended use could migrate over time or users can start using the software in a way that was not originally intended. Because of this shift, the deployed software is no longer in a validated state.

Each time a change is initiated to validated software, the intended use should be reviewed to ensure that it is still consistent with the actual use of the software.

5.5 Retirement phase

During the retirement phase, it should be documented the decommissioning of the software and to establish methods for accessing any associated electronic records throughout any required record retention periods.

Software retirement activities are highly dependent on the type of software being retired. Some software simply performs an activity and does not store any data. Other software can be as complex as a lot traceability or document control system, which houses volumes of product-related and compliance-related data. In the instance of software that stores data, a plan should exist for handling the data. Some issues to consider include the following.

- Is there software replacing the retired software?
- Can the data be migrated to the new software?
- Should the data be migrated to a portable format for long-term retention?
- What are the data retention requirements for the type of data?
- Will the data be stored on durable media?
- If so, what are the storage instructions or procedures and can the data be retrieved containing all associated data requirements?
- What is the procedure for maintaining durable media and software that can read it?
- Will an archived hardware platform be stored for using and retrieving the retired application?
- How will stored hardware be maintained?
- Would the retired software ever need to be accessed as part of a complaint or CAPA investigation?
- Will the platform and application be needed to re-create a software program?

6 Documentation

It should be ensured that all information associated with the software life-cycle control activities is appropriately documented and controlled.

Two major benefits arise from having high-quality and efficient documentation.

- a) Complete software definition that is clearly articulated in documentation enables full understanding of the software's intended use and expected performance and it enables the understanding of the full impact of any and all changes made to the software.
- b) Record of validation planning and execution provides documented evidence of the decisions made as a result of critical thinking. Focusing this documentation around the evaluations or analyses performed and the resulting tool selections that target risk-based and meaningful confidence-building activities provides for a succinct understanding of the validation that was performed. With a summary of how the acceptance criteria were met, the documentation provides evidence that the activities completed ensure the software performs as intended and introduces acceptable levels of risk to the process it automates.

The extent of documentation produced is directly related to the level of effort applied to the validation of the software. The level of effort should be commensurate with risk. Therefore, the software validation approach discussed in this document bases the extent of documentation on the impact of the process failure. The greater the risk of harm to persons or to the environment posed by the process, the greater the extent of documentation expected. In addition, the higher risk of harm should drive a higher level of scrutiny of the documentation by multiple cross-functional peers, by higher levels of management within the company or by both.

The organization of the life-cycle control information into documentation can vary depending on many factors, such as the technology used and the size or complexity of the software.

The information should be organized in a manner that facilitates the auditing of the information along with the ability to maintain evidence of a validated state during the maintain phase of the software life cycle.

How life-cycle control information is captured and documented depends on preferences and established policies of the parties performing the validation. Discretion is given to the parties validating the software regarding how the objective evidence of life-cycle controls is packaged and presented in documentation. From a compliance review perspective, the validation planning and reporting documentation should be established to provide a compilation of all value-added, confidence-building activities that were planned and executed in order to ensure that the software performs as intended. Essentially, this documentation is the key record of the choices made (decisions) on the basis of inputs (decision drivers) that embody the critical thinking process used to confirm that a complete software solution has been developed that meets the intent of the regulation and considers all key stakeholders and their needs.

NOTE The term “documentation” is used to refer to the body of information that is recorded, whether it is recorded in an actual document or in tools that capture the information, such as requirements management tools.

7 Prerequisite processes

The methodology presented in this document is intended to operate fully within an effective quality management system in order to be more effective.

The aspects of a quality system that can have the most positive effect on the success of the critical thinking methodology include asset and infrastructure management (human and hardware), change management (including configuration management) and vendor management. Detailing those aspects is outside the scope of this document; each aspect is addressed in other standards and documents within industry (see Bibliography). In addition, this document does not intend to associate specific roles or functions (e.g. quality assurance, management and manufacturing) with the activities in this document. Each company's philosophy and human resource infrastructure will dictate the acceptable roles for performing validation activities.

Annex A **(informative)**

Toolbox

A.1 General

This toolbox provides a list of confidence-building activities that can be conducted to satisfy the intent of the requirement with regard to validation. It is not meant to be an exhaustive list of available activities for this purpose, but it provides a starter set based on the current software engineering body of knowledge. Some of these activities overlap or work together, for example, normal case testing is often a part of software system testing, but the focus here is on the value of the activity. The activities are to be used as a foundation for validation planning and execution.

Selection and conduction of activities should be appropriate for the risk associated with the software. To support this selection, the activities in the toolbox are categorized and marked as such according to the following scheme.

- Full extent: Conduct this activity as is in any case.
- Tailor: Select and conduct the appropriate parts of this activity.
- Select: Select and conduct activity where appropriate.

A toolbox can be customized to define the activities used in your organization and can evolve over time as technology changes and as lessons are learned, thus incorporating new software engineering best practices. Where applicable, some of the activities would also be procedurally called out in standard procedures.

A.2 Toolbox structure

The activities are organized for convenience into five main software life-cycle process activities. Depending on the scope and nature of the software, critical thinking should be applied at various stages in the software life cycle to identify and select the activity most appropriate to the software.

For each named activity appearing in the list, there is a short definition and a description of the value that the activity contributes to the validation effort. The definition column also contains examples of methods one can use to accomplish the named activity.

Table A.1 — Development phase: Define

Activity	Definition
Process requirements definition (Full extent)	Activity for the definition of the process under consideration for partial or full automation by software, a manufacturing process or a quality system process. Activity also describes any verification or preventive measures within the process that can be considered when performing process or software risk analysis. The output from this activity can be documented in a process flow schematic or requirement statements that define the activities performed within the business, manufacturing or quality system process.
Process failure risk analysis (Full extent)	Activity for determination of the impact of process failure on device safety and efficacy, manufacturing personnel, the environment or the quality system.
Intended use (Tailor)	Activity for simple software can consist of a few sentences or paragraphs. For large, complex software, activity can include extensive documentation across multiple documents and might include detailed software requirements. Risk is also an important factor in determining the depth of the intended use definition. Elements of intended use: — software purpose and intent; — software use requirements; — software requirements.
Validation planning (Full extent)	Validation planning is performed in two phases: — during the develop-define phase to define the level of detail and effort expected in validation documentation, the level of scrutiny and to select the activities to be included in the define phase; — later, during the implementation phase, to select the appropriate validation activities on the basis of decisions made during the define phase and associated risk analysis activities. The output from validation planning is a plan that describes the activities that will be performed to establish confidence that the software consistently meets the requirements of its intended use.
Formal software requirements review (Select)	Activity (process, meeting, etc.) in which stakeholders review and agree on software requirements based on intended use.
Software development life-cycle model choice (Select)	Activity for the definition of the life-cycle methodology and controls to be used during the development portion of the total software life cycle. Usually needed only for complex or risky software. IEC 62304:2006/AMD1:2015 might be particularly appropriate as a process standard for some software.
Risk management planning (Full extent)	Activity related to planning how the risk management for the software will be performed. The output of risk management planning is a plan defining an approach for analysis of areas of concern for the software relative to risk and a choice of methods by which to analyse risks, such as failure modes and effects analysis (FMEA), fault tree analysis or other tools.
Identification of risk control measures within the manufacturing or business process (Full extent)	This activity is a mechanism to identify measures to control risks or hazards (e.g. procedural controls). It includes continuous monitoring to make sure that the controls are in place and working properly.

Table A.2 — Development phase: Implement

Activity	Definition
Analysis of software failure (risk analysis) (Full extent)	Analysis of software failure refers to the determination of the impact of software failures relative to the process and to the areas of concern identified in the analysis of process failures.
Software architecture documentation and review (Select)	Software architecture defines the high-level structure of the software elements of software and the relationship between them, documenting the architecture and reviewing for correctness, completeness and ability to perform software functions.
Design specification (Select)	Design specification is a precise statement of how the software requirements will be implemented. It typically includes software or component structure, algorithms, control logic, data structures, data set use information, input and output formats, interface descriptions and the like.
Development and design review (Select)	Development and design review is a review conducted to evaluate the progress, technical adequacy and risk resolution of the selected design approach for one or more configuration items.
Identification of risk control measures within the software design (Full extent)	This activity identifies measures to control risks or hazards that were identified during the risk assessment. Identification of risk control measures should be an iterative process to allow continuous monitoring and to ensure that the controls are in place and working properly (e.g. procedural controls, hardware redundancy).
Code review or code verification (Select)	Code review or code verification consists of a peer review of software source code intended to find and remove defects and improve overall code quality. Code reviews and overall code quality can be enhanced by establishing and adhering to a set of common coding standards.
Traceability analysis (Select)	Traceability analysis refers to traceability of requirements to design, to code, to testing, to risk or hazard analysis and to risk control measures. It might also include traceability to process requirements.
Vendor audit (Select)	Vendor audit means assessment of software vendor systems to the level necessary to ensure the purchaser that the vendor is sufficiently capable of supplying safe and usable software. A variety of vendor audit methods are possible.

Table A.3 — Development phase: Test

Activity	Definition
Test planning (Select)	Test planning should define the overall approach to the testing activities that help build confidence that the software meets its intended use. However, software testing by itself might not be sufficient to establish confidence that the software is fit for its intended use. Other verification techniques might need to be combined with testing to ensure a comprehensive validation approach. The level of testing should be based on the risk drivers and factors and should provide the appropriate level of confidence to demonstrate that the software meets the requirements and design specifications in accordance with the appropriate testing methods. Such testing can include developer testing, unit testing, integration testing, user testing, load testing, operational testing and the like.
Unit testing (Select)	Testing conducted to verify the implementation of the design for one software element (e.g. a unit or module) or a collection of software elements.
Data verification (Select)	Data verification refers to activities completed to confirm the correctness of data. It might be done as part of a data migration, conversion, or testing effort or independently, and it can include statistical sampling where appropriate.
Integration testing (Select)	Integration testing is an orderly progression of testing in which software elements, hardware elements, or both are combined and tested to evaluate their interactions until the software has been integrated.

Table A.3 (continued)

Activity	Definition
Use case testing (Select)	Use case testing is a form of functional testing that ignores the internal mechanism or structure of a system or component and focuses on the outputs generated in response to selected inputs and execution conditions. Each use case can have input parameters associated with it and each parameter can have a set of values identified to simulate actual use conditions. A series of use cases can be connected using predetermined flows that describe a sequence that accomplishes some goal.
Interface testing (Select)	Interface testing refers to the confirmation of the interface between software applications, taking into account the entire data transfer path from output to input. Interface testing can be accomplished through direct testing or 100 % data verification. Testing activities should include strategies that ensure that the interface performs as required at the specification limits or at boundary conditions for both normal and abnormal cases.
Regression testing (Select)	Rerunning test cases that a program has previously executed correctly in order to detect errors spawned by changes or corrections made during software development and maintenance.
Vendor-supplied test suite (Select)	Vendor-supplied test suites can test the full capability of a software solution and can provide significant confidence in the performance of the software in the end-use environment. However, such suites should be assessed for appropriateness to the defined intended use and completeness of the testing, including testing for any risk control measures in place. Use of such a suite could require a contractual agreement requiring the vendor to maintain the test suite for the life of the software.
Software system testing (Select)	Software system testing is the process of testing an integrated hardware and software system to verify that the software meets its specified requirements. Such testing can be conducted in both the development environment and the target environment. Software validation differs from software system testing because software validation verifies the suitability of the software 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. For production systems controlled by software, process validation testing can cover some or all of these tests. For quality systems applications, performing all the steps required by the software work instruction could cover the software test requirements.
Use case testing (Select)	Use case testing refers to testing performed on the basis of use cases, including alternative flows and error conditions defined in those use cases.
Normal case testing (Select)	Normal case testing is testing with usual inputs.
Robustness testing (stress testing) (Select)	Robustness testing should demonstrate that a software product behaves correctly when given unexpected, invalid inputs. It is conducted to evaluate a system or component at or beyond the limits of its specified requirements. Methods for identifying a sufficient set of such test cases include equivalence class partitioning, boundary value analysis and special case identification (error guessing).
Output forcing testing (Select)	Output forcing testing means choosing test inputs to ensure that selected (or all) outputs are properly generated by the system. Output forcing involves making a set of test cases designed to produce a particular output from the system. The focus is on creating the desired output, not on the input that initiated the system response.