TECHNICAL REPORT

ISO/TR 80002-2

First edition
2017-06

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)

Contents

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.

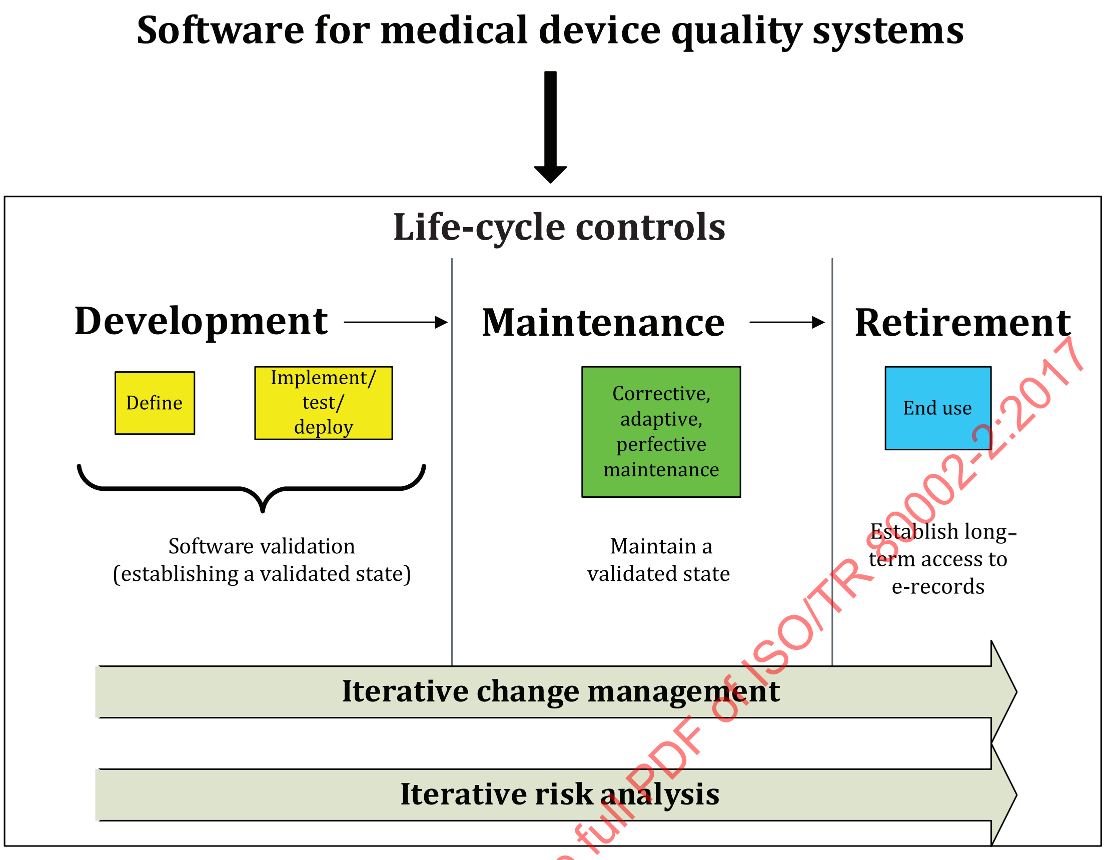

Software for medical device quality systems
Life-cycle controls
Development
Maintenance
Retirement
Define
Implement/ test/ deploy
Software validation (establishing a validated state)
Corrective, adaptive, perfective maintenance
Maintain a validated state
End use
Establish long-term access to e-records
Iterative change management
Iterative risk analysis

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.

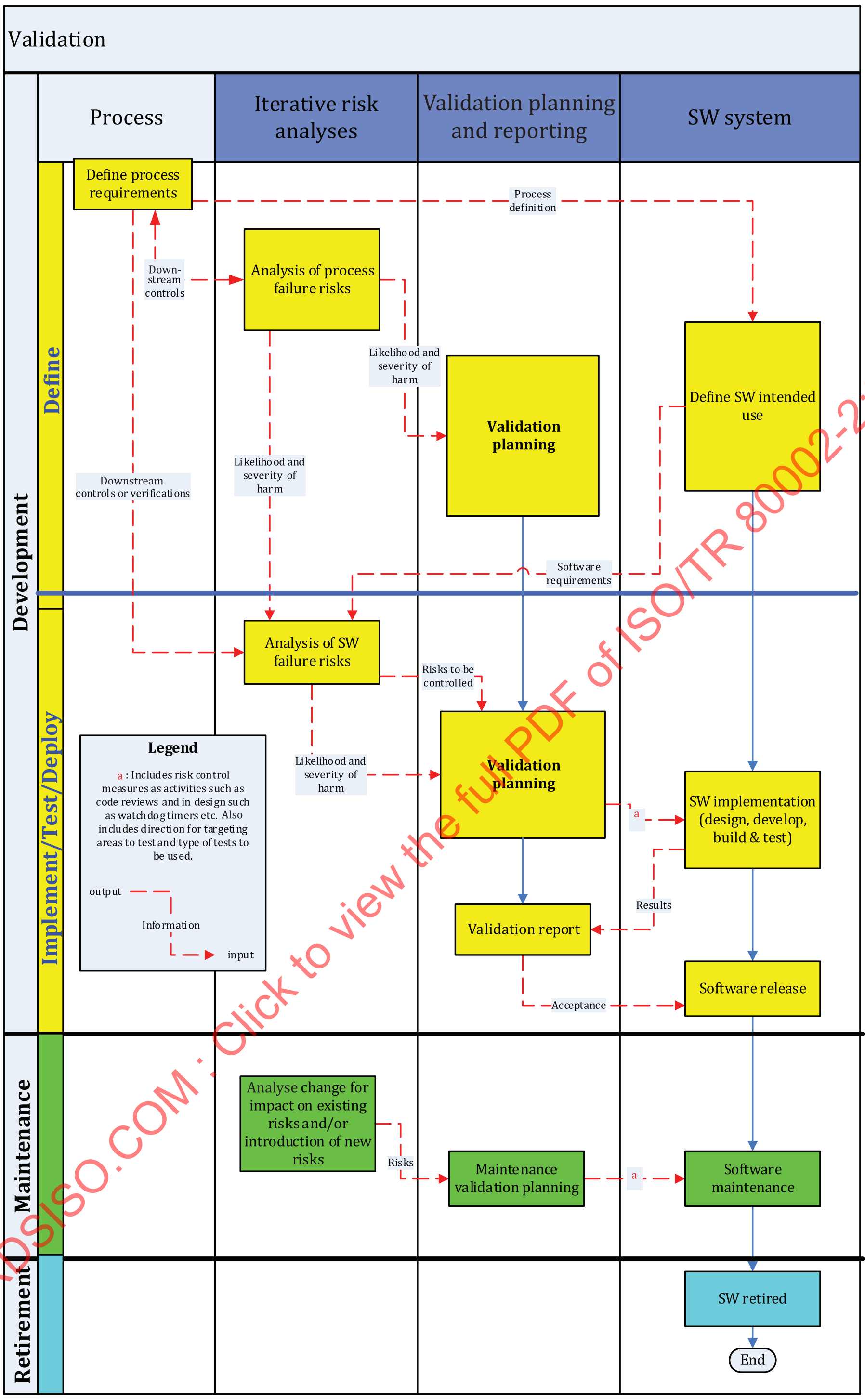
During the development phase of the life cycle, risk management and validation planning tasks are used to define the appropriate level of effort to be applied to the software and to determine what confidence-building tools to apply. This type of approach results in the completion of appropriate value-added 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.



Validation
Process
Iterative risk analyses
Validation planning and reporting
SW system
Development
Define
Implement/Test/Deploy
Maintenance
Retirement
Define process requirements
Process definition
Down-stream controls
Analysis of process failure risks
Likelihood and severity of harm
Validation planning
Define SW intended use
Downstream controls or verifications
Likelihood and severity of harm
Software requirements
Analysis of SW failure risks
Risks to be controlled
Legend
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.
output
Information
input
Likelihood and severity of harm
Validation planning
a
SW implementation (design, develop, build & test)
Results
Validation report
Software release
Acceptance
Analyse change for impact on existing risks and/or introduction of new risks
Risks
Maintenance validation planning
a
Software maintenance
SW retired
End

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.

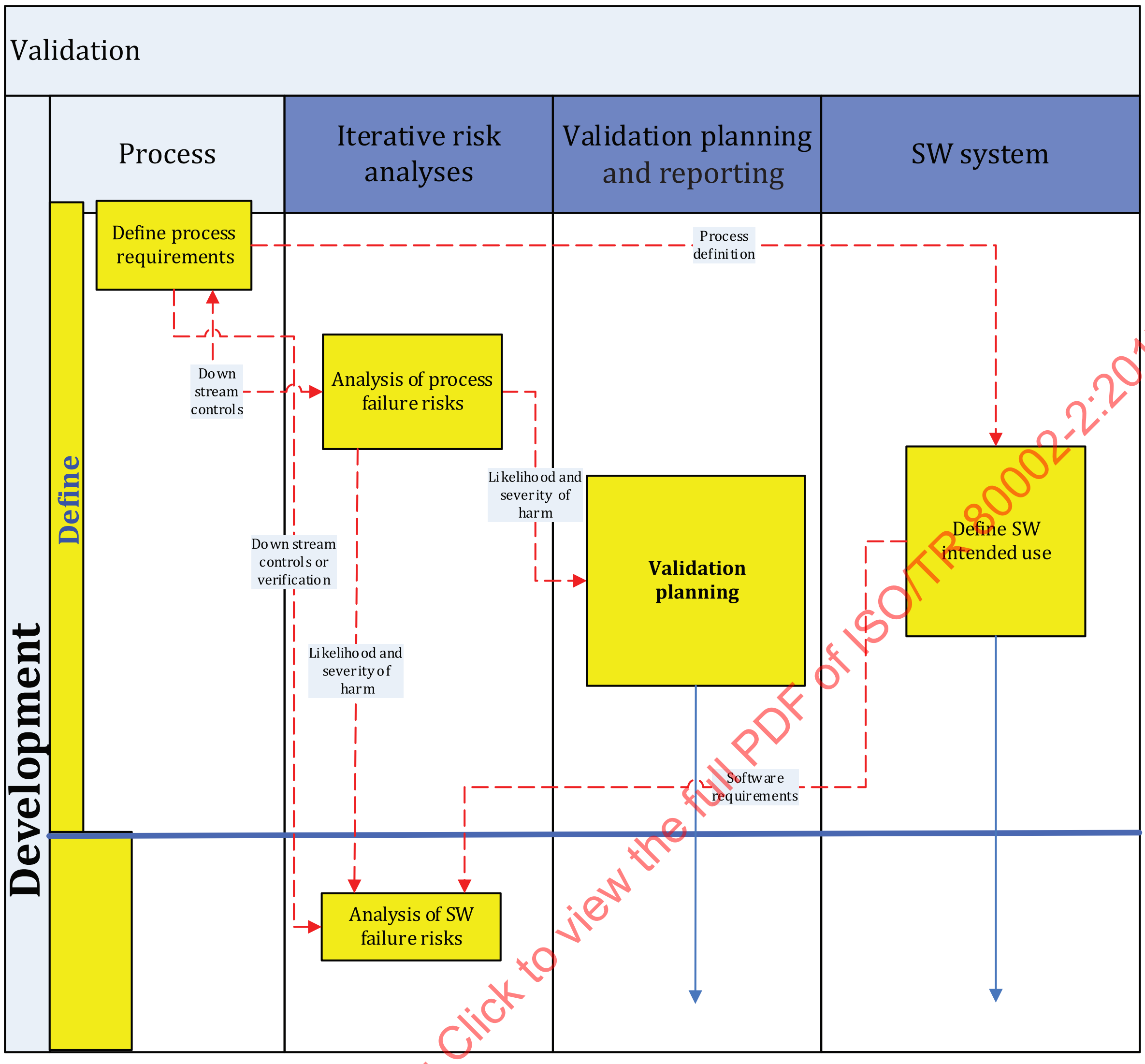

Validation
Development
Define
Process
Iterative risk analyses
Validation planning and reporting
SW system
Define process requirements
Process definition
Down stream controls
Analysis of process failure risks
Likelihood and severity of harm
Validation planning
Define SW intended use
Down stream controls or verification
Likelihood and severity of harm
Software requirements
Analysis of SW failure risks

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

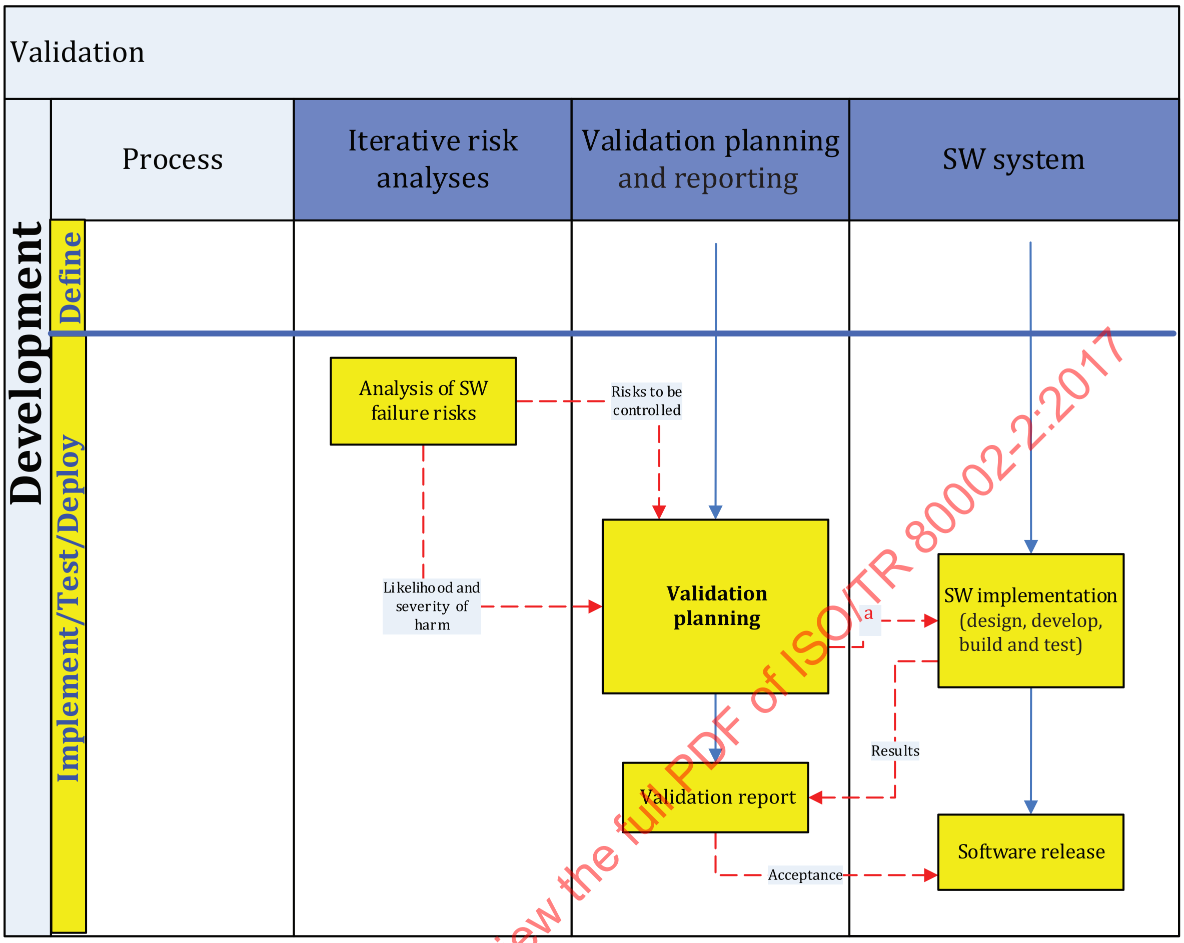
This activity uses the intended use definition and the results of the software risk analysis as inputs to the identification of risk control measures and the selection of tools from the toolbox that will be used to validate the software.

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.

Table A.3 *(continued)*

Activity	Definition
Combination of inputs testing (Select)	Combination of inputs testing is a testing technique by which a combination of inputs that a software unit or system might encounter during operation is exercised.
Beta testing (Select)	Beta testing is testing by the vendor in a live environment for a small set of clients.
Performance testing (Select)	Performance testing measures how well the software system executes in accordance with its required response times, central processing unit (CPU) usage and other quantified features in operation.

Table A.4 — Development phase: Deploy

Activity	Definition
User procedure review (Select)	User procedure review is the review of the user procedures and instructions related to the use of the software. Such a review ensures that the use of the software is properly defined.
Internal training for the application (Select)	Internal training refers to documented training activities specific to the software.
Installation qualification (Select)	Installation qualification means establishing confidence that the software is installed and functioning according to the documented installation instructions.
Operational and performance qualification (when process validation is performed) (Select)	Operational qualification establishes confidence that the manufacturing process and associated systems are capable of consistently operating within established limits and tolerances. Performance qualification establishes effectiveness and reproducibility of the process.
Final acceptance testing (Select)	Final acceptance testing refers to tests applied to the system just before final deployment. It is also known as go-live testing.
Operator certification (Select)	Operator certification is confirmation that trained individuals show evidence of competence in the training.

Table A.5 — Maintain phase

Activity	Definition
Maintenance planning (Tailor)	Methods associated with maintenance planning are as follows. Forward planning. This method covers the forward planning and anticipation of changes to the software. This method can be used during the initial implementation of the software before entering the maintain phase, but it also can be used at any time during the maintain phase. Planning for pending changes. This method covers planning done when a change to the software is pending. The planning typically focuses on activities specific to the pending change. This planning is done during the maintain phase of the software.
Known issues analysis (Select)	Known issue analysis is a process by which any and all issues with the software that are known by the vendor are assessed as to their impact to the use or validated state of the installed software.
Compatibility testing (Select)	Compatibility testing is the process of determining the ability of two or more software systems to exchange information.
Infrastructure compatibility analysis (Select)	Infrastructure compatibility analysis is the process of determining how changes to software infrastructure can affect the installed software. These changes could include changes to hardware or to the location of the system.

Table A.5 *(continued)*

Activity	Definition
System monitoring (Select)	System monitoring includes techniques used to evaluate the general health of the software system during the maintain phase of the software life cycle. Methods for system monitoring can include the following: — periodic assessment of whether intended use has changed; — actual use by end users; — training effectiveness evaluation; — defect analysis; — data auditing.
Backup and recovery processes (Select)	Backup and recovery processes include system backups, storage and retention of backed-up media, and recovery procedures for restoring data from backup media.

Operational controls
(Select)
	In addition to backup and recovery processes, monitoring and reporting, operational controls can be used to help ensure the software is operating as intended. Common methods include the following:
	— security;
	— access rights administration;
	— database administration;
	— archiving;
	— contingency planning.

Regression analysis (Select)	Regression analysis includes tasks such as traceability analysis or impact analysis. It is conducted to determine the required activities for maintaining the validated state of the system.

Annex B
(informative)

Risk management and risk-based approach

B.1 General

As mentioned in the core part of the document, the content and rigor of validation is determined by the risk associated with the software.

In order to expand on this concept refer to ISO 14971. ISO 14971 describes a risk management process to be applied for medical devices. However, the underlying principles, as well as the terminology, can be applied to the software that is subject to ISO 14971.

B.2 Terminology

The definitions listed below have either been taken from ISO 14971 or are based on the definitions in ISO 14971.

— hazard: potential source of harm;

— hazardous situation: circumstance in which people, property or the environment are exposed to one or more hazard(s);

— risk: combination of the probability of occurrence of harm and the severity of that harm;

— harm: physical injury or damage to the health of people or damage to property or the environment;

— severity: measure of the possible consequences of a hazard;

— risk control measure: measures by which risks are reduced to, or maintained within, specified levels.

B.3 Basic principle

The basic principle is to reduce the risks associated with the software to an acceptable level. In order to accomplish this, the manufacturer needs to identify possible hazardous situations with the use of the software, estimate the associated risks and evaluate if these risks meet acceptance criteria which, if not given otherwise, e.g. by regulations, are defined by the manufacturer.

Especially as software cannot harm on its own, the whole process that is controlled by the software is subject to risk management.

B.4 Identification of hazardous situations and estimation risks

Following the approach of ISO 14971 starting with intended use, the possible hazards and hazardous situations should be identified and the associated risks should be estimated. However, the possible harm to consider is quite different from that under consideration in ISO 14971.

Failure of production and quality system seldom results in direct harm to a patient or user of the medical device whose manufacture or quality is controlled by software. The harm in this context is almost always indirect. It is the harm to the device that ultimately becomes a source of harm to the patient or user of the device. This is not to say that indirect harm is in any way less severe. In fact, in some ways, the severity of failure of production and quality systems could be considered to be more severe simply because a single failure in these systems could lead to failures in many devices, ultimately harming many patients before it is detected. A failure of software in a single device can harm only one patient at a time.

Both direct and indirect multiple harms can result from failure of production or quality systems. Note that the harms in the list that follows are not mutually exclusive. Each has the potential for indirect harm to patients or users of a medical device. Examples include, but are not limited to, the following:

- harm to the medical device:
  - a machine tool does not produce a critical tolerance;
  - a calibration system miscalibrates a medication delivery device;
  - a sterilizer controller fails in a way that non-sterile components are produced;
  - failure of a final test system does not detect latent device flaws;
- harm to the manufacturing process:
  - a failure of a process controlled by software slows production rates as manual workarounds are used;
  - a failure of software-driven processes creates a high percentage of out-of-tolerance parts;
- harm to regulatory compliance:
  - a complaint-handling system misreports failure statistics, thus allowing field-reported defects to go undetected;
  - a device service or repair system fails to highlight trends for issues that could point to previously undetected defects;
  - a loss of integrity occurs to a database for implanted devices;
  - a loss of quality control records relating to safety checks on manufactured items occurs;
  - a loss of compliance data occurs;
  - a loss of device validation data occurs;
  - an inability to control and report the configuration of software in manufactured devices happens;
  - a failure of an MRP system to provide traceability results in failure to notify potential users of device safety recalls;
- harm to manufacturing personnel or the environment:
  - an operator is injured;
  - toxic chemicals are released.

All categories of harm should be considered in analysing the risks associated with depending on software to automate production and quality systems.

Estimation of risk comprises of the estimation of the severity of the possible harm and the likelihood of occurrence of that harm.

Estimation of severity is usually done by a classification (see e.g. ISO 14971:2007, Annex D or G.4 which connect to the acceptance level (see B.5).

It might turn out that it is difficult to estimate the likelihood of harm especially when looking at likelihood of software faults contributing to the harm. In this case, it should be kept in mind that a software fault is only one factor leading to a harm and several other factors outside the software might be involved (sequence of events). It is useful to assume a worst case for unknown likelihoods of events and finally a worst case of likelihood of harm.

A similar approach is taken by IEC 62304:2006/AMD1:2015. As the basis for the decision about the rigor of process controls, it assumes a worst case likelihood of faults of the software but allows for consideration of lower likelihood of harm associated with a sequence of events beyond the software (see also IEC/TR 80002-1).

B.5 Risk evaluation

Once the risks have been estimated, they need to be evaluated to see if they are acceptable or not. If not, the manufacturer should identify and implement risk control measures to reduce the risk to an acceptable level.

Perhaps the most difficult activity in risk management is determining what is an acceptable level of risk. Such a determination is highly dependent on the severity of potential harm. Each manufacturer needs to establish criteria for defining and documenting the acceptability of a risk and for identifying all risks in a format that will allow for evaluation of conformance to those criteria. In general, if an acceptable risk is reduced to a level that one is comfortable in defending to one's peers, management or auditors, then the risk is probably set at an appropriate level.

It is beyond the scope of this document to recommend acceptability thresholds, but a few recommendations on the process of setting them are appropriate.

— Be specific. Acceptance criteria such as "as low as possible" or "as safe as any other product" are not useful. Acceptance criteria should read like testable specifications, so that it is possible to objectively determine if the criteria for acceptability have been met.

— Where it is difficult to estimate the likelihood of harm, the acceptance criteria can be based on severity only.

— Acceptance criteria can relate to predefined selection of software process controls (i.e. selection of tools listed in Table A.1 to Table A.5).

— Identify acceptance criteria early. Set the goal or specification as soon as the potential risk of harm is identified. It is important to set the acceptability goals before any attempt is made to control the risk. The perception of acceptability often migrates to higher risk levels once an attempt has been made to control the risk. Documenting acceptability criteria in advance keeps the process from migrating.

— Document your rationale for determining the acceptability of risks. Such documentation is useful for future maintenance of the process and for communication of the thought process to regulatory investigators.

B.6 Risk control

B.6.1 Unacceptable risk

If the risk has been evaluated as not acceptable, the manufacturer should identify and implement risk control measures to reduce the risk to an acceptable level. These risk control measures can affect the software or other parts of the process.

B.6.2 Risk control measures not affecting the software

Examples of risk control measures not affecting the software are procedural changes, hardware redundancy, backup systems, monitoring systems, output verification (downstream verification) or vendor inspections.

Often, embedded production process software is difficult to access and few details are available from the manufacturer. A common example is software embedded in a machine tool that is used in the fabrication of a medical device. Validation of this type of software for intended use can be difficult when the software is validated on a stand-alone basis.

A risk control measure that works particularly well in these situations is downstream verification of the outputs of the software or of the outputs of the device controlled by the software. In other words, one can directly determine the suitability of the software for its intended use by monitoring the outputs of the software-controlled process for any and all potentially harmful defects. Such an approach could substitute for inferring the suitability of the software for its intended use by applying life-cycle control methodologies. This methodology is practical only for processes that have a reasonably small number of critical operations that can be checked on each part or on a statistically determined sampling of parts. The validation engineer should detail the rationale for substituting downstream verifications and any assumptions used to justify choosing sampled verification over continuous verification, and those assumptions should then be tested.

Downstream verification should be documented just as any other risk control measure would be documented. In particular, it is important to document that the verification process is a risk control measure so that it is not eliminated in cost-cutting measures at a later time. Furthermore, downstream verification results should be documented because the definition of validation require "provision of objective evidence" of validation, and this verification step substitutes for a large part of the validation. As the product evolves, the intended use of the software-controlled process can also evolve. As an example, consider a machine tool that initially performed one critical operation on a component of a medical device. Later, the medical device design was slightly modified in such a way that two critical operations were required of the software-driven machine tool. The intended use of the machine tool changed (two safety-critical operations versus one safety-critical operation), and consequently the downstream verification should change to verify both operations.

Downstream verification can be accomplished by manual operations or other human operations. Examples might include visual inspections for burred edges or mechanical alignments and manual measurements for mechanical tolerances or electrical continuity. Regardless of the nature of the test, if it is a downstream verification of a software-controlled process and if it is used as a risk control measure for that process, then the verification test should be documented. The test procedure for the human tester should be detailed with clearly defined passing and failing ranges of results for each parameter tested. The testers should also provide documented evidence that they have executed the procedures to test the process outputs.

B.6.3 Risk control measures affecting the software

Risk control measures affecting the software are either

— design changes, or

— process controls.

In the context of this document, selection of process controls is also referred to as rigor of validation and means selection of tools defined in Table A.1 to Table A.5.

Preferably, well-understood risk control measures outside the software, e.g., "downstream verification", as well as software design changes, should be implemented before just relying on process controls. However, a minimum set of process controls should be applied, especially in order to provide confidence in proper implementation of software design changes as risk control measures.

B.6.4 Verification of risk control measures and evaluation of residual risk

The implementation of risk control measures should be verified. Risk control measures beyond process controls should be verified for effectiveness. In this case, the residual risk should be evaluated for acceptability.

Annex C
(informative)

Examples

This document applies to software used to automate parts of quality systems and manufacturing processes, including the generation, measurement, assessment or management of data intended for regulatory submission, quality system, production and data processing. Other intended uses might include the direct or indirect capture of data from instruments, operation and control of equipment and the processing, reporting and storage of data. For those different activities, software can vary from software contained in a programmable logic controller (PLC) or a personal computer (PC) to software contained in a laboratory information management system (LIMS) with multiple functions. The following are some examples of intended uses:

— software that makes pass/fail decisions on product;

— software used for custom record-keeping within the quality system;

— data manipulation and analysis software used for product submissions;

— data manipulation and analysis software used for reporting to regulatory agencies;

— any software development tools or compilers used for regulated-process software;

— any software tool or subordinate software tool responsible for qualifying and verifying life-critical software;

— any software used for component, product or patient traceability within the quality system;

— any “software of unknown origin” (i.e. no knowledge of the quality and robustness of the software is available) used for the above-mentioned purposes.

The examples presented in this annex represent an attempt by the authors of this document to offer practical, realistic examples of software that a medical products manufacturer might encounter. The best way to experience the critical thinking approach and to appreciate the variability across software types, software risks and intended uses is to offer these examples.

Note the following qualifiers.

— The examples used here include the results of critical thinking as performed by the authors of this document and represent an acceptable level of validation effort and rigor that will add value and provide confidence that software will function as intended. Readers are strongly encouraged to consider what activities and level of effort make sense from an engineering perspective, as well as to determine the required rigor based on the key factors for software used for medical device quality management system processes.

— There is always more than one way to establish confidence in the appropriateness of the validation effort. The examples presented in this document provide a methods-based approach that is based on current thinking and the experience of the authors of this document.

— Readers are strongly encouraged to view the authors’ efforts as neither authoritative nor prescriptive. The examples cited are similar in format only for the presentation of the data and include key thought processes to demonstrate the use of critical thinking. This layout is not intended for use as a validation template, nor does it contain all the depth and detail that would be expected for actual validation documentation.

- The examples used assume that the prerequisite processes identified in Clause 6 are present and are in good working order. Although the examples do not contain extensive reference to the prerequisite processes, those processes should be in place to ensure that the software and all associated aspects, such as documentation and other infrastructure, are subject to change control.
- Each example begins by clearly defining the process to be controlled. Therefore, it has already been determined that the process and, hence, the software are in scope. Critical thinking activities are then identified and summarized.
- The examples used here are meant to provide information about the decisions and drivers of decisions used in the critical thinking process and do not necessarily represent the comprehensive validation of the software discussed.
- Any company names, teams or individuals used in the examples are purely fictitious and are included only to facilitate the discussion.

The examples used here are generally focused on bringing a particular system into a validated state. Although establishing a validated state for a system is of great importance, maintaining the validated state during the maintenance phase of the system is also vital to ensuring the proper operation of software and surrounding processes. Maintenance activities require the same controls and critical thinking as are required by the initial validation activities.

Example 1: Programmable logic controller (PLC) for manufacturing equipment

Background

The Tubing Supply Company has been contracted to supply a major medical device manufacturer with tubing for its intravenous (IV) systems. The company has received the specifications for the tubing, including requirements for the tubing to be formed into a proprietary shape. This special tube-shaping requirement will be performed at the Tubing Supply Company as part of the manufacturing process for its tubing segments.

This tubing formation process is of particular concern to the supplier because the tubing formation is a unique process that the supplier does not currently have the machinery to perform. It is decided that a customized piece of equipment with a programmable logic controller will be developed to perform this task. This equipment and the PLC contained in it should be validated for their intended uses, as required by the medical device company's policies.

Defining the process

The tubing supply company and the device manufacturer establish a team of people to define the process by which the tube will be formed. The process defined in the meeting uses temperature and pressure to form a shape in a piece of plastic tube. The steps include the following:

a) obtain materials;

b) insert into machine;

c) deform tube to proper diameter through pressure and heat;

d) allow cooling of tube;

e) remove tube from machine;

f) measure tube for proper diameter.

Analysing the process risk

The medical device manufacturer has communicated to the Tubing Supply Company that the following issues and associated hazards arose from the risk analysis process.

— Lack of good connection to the fluid bag results in a leak that is not hazardous but could have a risk of caregiver slippage. Leakage could also delay treatment.

— Cosmetic issues could affect customer acceptance and cause delay of treatment.

— Potential exists for the operator to be burned during the tube-forming process.

Prior to mitigation, there is moderate level of risk associated with product failure because of the hazards of caregiver slippage, delay of treatment and operator burns.

The following process risk control measures are currently in place:

— upstream operations, such as incoming inspection and line clearance, to ensure that the tubing is acceptable for use;

— downstream verification checks, including leak testing, in-process inspection and test fittings, that mitigate equipment error;

— a shield, an independent temperature sensor and a coolant sprayer put in place to prevent operator injury.

Using this information, the supplier works with the medical device manufacturer to conclude that there is a low residual risk of failure of the tubing as a result of the tube-forming process.

Defining the software purpose and intent

The Tubing Supply Company knows that to validate software for its intended use, intended use should first be defined. To achieve consensus on what the equipment is meant to do, the team members ask themselves a series of questions intended to determine a succinct but usable definition of the system's purpose and intent. They produce the following statement.

— The software-controlled equipment is intended to automate steps 2 to 6 of the defined process. The system is intended for use in facility B, manufacturing line 3, for the creation of PN 001. The system will automate the insertion, forming, removal and measurement of tubing for an IV for the delivery of general, non-hazardous solutions.

Validation planning

The first step in planning for validation involves determining the rigor and review of deliverables. Because the residual process risk was determined to be low, the following approach was taken.

— Documentation rigor:

  — The documentation in this project will have medium rigor, meaning that there will be instances when deliverables will be combined, and designs will not be translated to detailed design specifications before implementation.

— Level of scrutiny:

  — Review and approval will be deliverable by those responsible for the development and implementation of the process (the Tubing Supply Company representative) and by an independent quality person (medical device company representative).

  — PLC code and all specifications/designs will be placed under formal configuration management, such as in a document control system or configuration control system.

— Defining the system:

  — Process requirements will be created and will include a system requirements specification that details the functionality of the equipment, including expected inputs and outputs of the equipment (e.g. design control elements for the entire functional piece of equipment).

  — The team will create an operator's manual for using the system from the operator's perspective. In addition, software requirements will be created and will include logical functional flow, which will be sufficient to cover the design on the software as well.

Establishing confidence and control over the software

Neither the Tubing Supply Company nor the medical device manufacturer has used this PLC programming package before. There is no history available for the Tubing Supply Company to help build confidence in the ability of the software to work as required. However, there will be control over the programming of the PLC through review of requirements, configuration control and testing of the system's functionality through test protocols.

Defining software boundaries with other systems

The PLC contains the only software in the piece of equipment. This software is not linked to any other system.

Software risk analysis

The software can fail by releasing a tube down the manufacturing line that has an incorrect shape, resulting in leakage and possibly in caregiver slippage. The software can also malfunction, resulting in excessive heat, which can lead to operator burns. The software itself does not introduce any new risks to the product that have not already been captured in the process risk analysis. Therefore, the group determines that the current downstream processes should remain and are sufficient to mitigate the risks associated with software failure.

Finishing the validation plan

Now that the team members know more about the software and its use, they should complete the validation plan as follows.

— Implementation tools:

  — A series of programmable parameters within the equipment include time, temperature and pressure. The desired settings and ranges for these parameters within the equipment are all captured in the software requirements. Therefore, the software requirements specification is sufficient for design purposes without additional design activities or documentation.

  — The team will establish a traceability matrix between the software requirements and their associated tests and will conduct a traceability analysis to ensure that the traceability is complete.

— Testing tools:

  — Software system testing will be based on the software requirements and procedures in the operator's manual.

  — Regression testing will be performed if needed.

— Deployment tools:

  — The system operators and engineers will review the work instructions for clarity and usability.

  — Use of the equipment will require operator certification.

  — After completion of the validation plan and execution of its activities, the team is comfortable that this system will consistently provide the desired and defined outputs.

Maintenance considerations

When changes are considered to any part of this process, or if there is a change to the intended use of the software, an analysis should be carried out to determine any current mitigations will be affected or if any new risks will be associated with the change. This analysis includes review of the software risks associated with the tube-forming equipment.

Toolbox usage

The following tools were used from the toolbox.

— Develop-define phase:

  — process requirements definition;

  — process failure risk analysis;

  — intended use;

  — validation planning;

  — software requirements definition;

  — identification of risk control measures within the manufacturing process.

— Develop-implement phase:

  — analysis of software failure;

  - traceability analysis.
- Develop-test phase:
  - software system test;
  - regression test.
- Develop-deploy phase:
  - user procedure review;
  - operator certification.

Example 2: Automated welding system

Dave is part of a team validating all the systems on a new manufacturing line. His job is to validate the case cover welder. For this project effort, he is the project manager.

Description of process

Dave's team spends a lot of time discussing who is developing and validating which parts of the new manufacturing line. When Dave gets the parts, they are already marked, and all of the materials are inspected and certified. The parts are tested on validated systems upstream.

To set up the welder, four steps are required:

— turning on the machine;

— confirming the presence of the bar code in the part to run;

— pulling the program for the part from the manufacturing execution system;

— confirming the proper program version against the device master record.

The case cover weld process itself has 10 steps:

a) opening the door;

b) loading the parts;

c) shutting the door;

d) starting the program;

e) placing vision system indexes at the start point;

f) turning on the laser;

g) ensuring that the motion control moves the part welds;

h) turning off the laser;

i) opening the door;

j) removing the part.

After this process is completed, the parts move to systems that are not Dave's responsibility. He knows that downstream activities include a destructive test of weld penetration, a height check on the size of the can and a leak check for hermetic seal.

Define intended use

To define the intended use for his software, Dave gathers information. He knows that accuracy of vision, motion, power and speed are all important to the process to protect the safety of the operator and to achieve consistent weld penetration.

Dave first defines his intended use by stating the purpose and intent of the software as follows.

— The software is intended to weld the case cover, protecting the machine operator from direct access to an operating laser. This includes steps e) to h) in the description of the process above.

Risk analysis

Dave would like to remove human error from the process. He knows that control of the laser, servomechanisms and vision are the key components of this process. The software begins by checking whether the door is closed. For safety reasons, the software will not start the process if it does not sense that the door is closed. The software ends by confirming that the laser is off and then by allowing the door to open. An emergency stop or an unexpected opening of the door cuts power to the laser. Dave uses information from the process and design risk management activities that occurred as part of the design of the process of which the weld is a part. He refers to the FMEA and focuses on three areas: critical part parameters, the hermetic seal and user interfaces. Dave identified multiple hazards related to this process. First, an operator could be burned if exposed to the laser. Related to the product, the process could improperly weld, resulting in a bad product that could leak and hurt the end user. Dave determines the risk of this process is high.

Validation planning

For this project, Dave looks at the define tools in the toolbox and determines that he needs to create a software requirements definition and maintenance document. His software requirements should include configuration parameters for tooling and laser time and power adjustments. He also needs to define the software to hardware interfaces. Specifically, Dave includes accuracy requirements for the vision system, laser time and power ranges, motion control accuracy requirements and door sensor safeguards, including interface to hardware door lock if the laser is activated.

Dave also determines that he needs to hold a formal software requirements review, which will include the automation engineer, the manufacturing engineer and the quality engineer.

The software for this system will be a purchased package, but Dave knows his company will need to make custom modifications. He needs to add an interface to the factory manufacturing execution system (MES).

Risk control measures

Dave next focuses on the risks. He sees the severity of the weld depth and other critical parameters as low because he is confident that the downstream leak check and the periodic destructive test to check weld penetration are sufficient. Similarly, the leak check will confirm that the hermetic seal is acceptable. This leaves the risk in the area of user interfaces and, specifically, the risk that the software could start the laser while the door is open. Dave is aware that there are software checks for the door seal, but because the severity of risk is high if the software fails to operate as intended, he adds a redundant hardware interlock to prevent laser activation with the door open.

Validation tasks

Next Dave turns to the validation tasks. The tool vendor that he selected has provided extensive programming tools. Therefore, the software requirements specification and review created earlier are sufficient for design without using additional design, development and configuration tools from the toolbox.

Another task that Dave has selected from the test section of the toolbox is test planning. The test plans are to include details of the software environments and the expected test results. The test plans need to be reviewed and approved by the automation engineer, the manufacturing engineer and the quality engineer, as well as by Dave. The test report will include the actual test results and compare them with the expected results, provide a pass/fail indication, include test identification and provide documentation of problem resolution and regression testing for any failures. For this report, Dave wants additional approval from the automation engineer, the manufacturing engineer, the quality engineer and the project sponsor.

Deployment

For the deployment of the welder, Dave reviews the deploy tools in the toolbox and decides that a manufacturing operator procedure is needed and that it needs to be reviewed by the automation engineer, the manufacturing engineer and the quality engineer. To ensure that the operator understands how to operate the welder, Dave creates an operator training and certification procedure that includes a test. He knows that the MES will not allow the operator to pull the weld program off the system without certification, so he is comfortable that the risk of injury to the operator has been successfully mitigated.

Maintenance

Dave knows that his firm has a configuration-checking tool. Therefore, no specific planning for maintenance is performed during this validation.

Example 3: Automated welding process control system

Table C.1 to Table C.14 in this example demonstrates the process steps illustrated in Figure 2.

Table C.1 — Example 3 — Process requirements

		Process	Iterative risk analyses	Validation planning and reporting	Software system
Devel-op	**Define**	**Process requirements (see 5.3.2.2)** Device Corporation is a class C (see GHTF/SG1/N77:2012) medical device manufacturer. Device Corporation has chosen to implement an automated welding process control system. To ensure that the device case is welded appropriately, Device Corporation will use a method that segregates products using a parametric release decision process. Device Corporation also has chosen to use the information from this process to support its device history record. Device Corporation has assigned a new project manager to validate the automated welding process control system. The project manager recognizes that this system needs to comply with the requirements of ISO 13485 for software validation. Therefore, the project manager recognizes that the proposed welding process control system requires validation. To better understand the requirements and risks involved in validating the welding system, the project manager defines the process as follows. a) The operator enters the lot number into the system for the first part of the lot. b) The operator inserts subcomponents into the machine fixturing. c) The operator presses the cycle start button. Fixturing is moved into mated position mechanically through hydraulics. d) The welding cycle starts in conjunction with a fixed speed rotation of the fixtured subcomponents. e) An infrared thermometer monitors the material temperature during the welding process. The temperatures are recorded in a file, along with the lot number and the part sequence number for each part welded. f) The machine opens the fixturing at the end of the cycle. g) The operator removes the welded part and places the part in a corresponding position in the lot tray according to the sequenced number. h) The operator repeats steps b) to g) until the lot tray is filled. i) The operator hits an end-of-lot button. j) The machine operator interface displays the part sequence numbers whose weld temperature is outside the process limits. k) The operator discards the corresponding part numbers from the lot tray. l) The operator prints the rejected parts list and sends the lot tray and report to the next station. m) The operator starts a new lot by repeating step a). The project manager also realizes that the key automation functions are as follows: — storing the lot number; — storing the weld temperatures per sequenced part number; — displaying the part sequence numbers that have exceeded the limits of the process temperature during welding; — printing the lot reject report.			

Table C.2 — Example 3 — Analysis of process failure risk

		Process	Iterative risk analyses	Validation planning and reporting	Software system
De-vel-op	De-fine	**Analysis of process failure risk (see 5.3.2.3)** The project manager then thinks about what could go wrong in the current process. The project manager realizes that if the process breaks down, release of improperly welded parts could expose patients to non-sterile devices. Accidental release of bad product could occur because of a welding process control system error or because of operator error. The project manager then considers what risk control measures are in place to mitigate the risk. The project manager learns that the Process Group has a procedure in place that verifies that the welding operator correctly rejected the parts at the next process step. Furthermore, the project manager learns that the welding system is a commercial OTS system.			

Table C.3 — Example 3 — Software purpose and intent

		Process	Iterative risk analyses	Validation planning and reporting	Software system
De-vel-op	De-fine	**Software purpose and intent (see 5.3.2.5.2)** With a basic understanding of the process, the project manager is ready to write the purpose and intent for the welding process control system. — The welding process control application makes closed-loop quality decisions as to the pass-or-fail status of welded cases. On the basis of these decisions, the welding operator manually rejects the non-conforming product. The project manager reviews the purpose and intent to appropriately capture the boundaries of the software within the process and decides to revise the statement as follows. — The welding process control application makes closed-loop quality assurance decisions as to the pass-or-fail status of welded cases. On the basis of these decisions, the welding operator then manually rejects the parametrically non-conforming cases. The welding station is the only control point in the entire device process that ensures device seal integrity. The project manager then considers what other systems, if any, will need to interface with the welding system. He determines that the software is a single application running on a PC connected to an infrared temperature device, an operator interface, a printer and a machine PLC input/output. The welding system is not connected to the network.			

Table C.4 — Example 3 — Validation planning

		Process	Iterative risk analyses	Validation planning and reporting	Software system
De-vel-op	De-fine	**Validation planning (see 5.3.2.4)** Now that the project manager understands the process and has determined the intended use of the new system, the project manager is ready to develop the validation plan at a high level. Earlier, the project manager determined that there is a high residual risk in the welding process because it is to be implemented as a non-verifiable process. Thus, the project manager determines that an extensive review of the validation effort is needed. The project manager decides that the key approval roles should be made by the Process Engineering and Quality Engineering Departments and by the operations process trainer. Moreover, the final product acceptance manager should approve the requirements. The project manager decides to start authoring the validation plan because the quality system requires that the validation plan be approved for high-risk systems before any other validation deliverables or project deliverables can be approved.			

Table C.5 — Example 3 — Software use requirements and software requirements

		Process	Iterative risk analyses	Validation planning and reporting	Software system
De-vel-op	De-fine	**Software use requirements and software requirements (see 5.3.2.5)** The project manager believes that it is necessary to provide a high level of detail or formality in this validation effort and knows that it is important to define detailed process and software requirements. The project manager now writes the software requirements. The project manager decides that the software should include redundancy in the temperature verification and reject decision process. The project manager also requires the system to be able to reprint the reject report any time before the occurrence of line clearance activities. Because this system supports parametric values, the project manager also includes security requirements, along with a detailed listing of what data values can be changed by system access level.			

Table C.6 — Example 3 — Analysis of software failure risks

		Process	Iterative risk analyses	Validation planning and reporting	Software system
De-vel-op	Im-ple-ment, test and de-ploy	**Analysis of software failure risks (see 5.3.3.2)** The project manager now needs to decide what approach should be used to establish full confidence in the welding system. The project manager notes that the welder design calls for a commercial off-the-shelf (COTS) system that is commonly used in industry. The project manager discovers that past problems or issues with this product have been quickly identified and publicized by the manufacturer. Although the project manager has already determined that welding process is of high risk, the project manager still wants to formally analyse the risk of a software failure. To confirm this intuition, the project manager reviews questions from the company's risk model. a) Is there a potential risk to product safety if the software malfunctions? *Yes* 1) How? *System accepts a bad part on the basis of default temperature limits. Limits reset to default setting after a power failure.* 2) What should be done to control this risk? *Require operator to verify the limits at the beginning and end of each lot run.* b) Is there a potential risk to product quality (other than a safety risk) if the user makes a mistake? *Yes* 1) How? *In manual mode, the welding laser can fire if both part sensors are triggered for 3 s.* 2) What should be done to control this risk? *Change the default configuration to fire only in auto mode.*			

Table C.7 — Example 3 — Validation planning

De-vel-op	Im-ple-ment, test and de-ploy	Process	Iterative risk analyses	Validation planning and reporting	Software system
		Validation planning (see 5.3.3.3) With an understanding of the software requirements, the project manager has enough information to finish the validation. The project manager has decided on the implementation approach and has analysed the software risk. At this point, the project manager steps back and asks this question, in light of everything learned about this system: “What validation activities would really allow me to gain confidence that the welding system is fit for its intended use.” The project manager thinks about how the system is being developed by a third party and is concerned that the developer correctly translate the requirements for the report customization. Because the system will depend on various data fields, the project manager adds a verification step activity in the code review to confirm the correctness of the developer’s work.			

Table C.8 — Example 3 — Software implementation

De-vel-op	Im-ple-ment, test and de-ploy	Process	Iterative risk analyses	Validation planning and reporting	Software system
		Software implementation (design, development, building and test) (see 5.3.3.4) The decision to purchase rather than internally develop the software was made on the basis of the availability of a commercial off-the-shelf (COTS) system. However, the project manager still needs to prove to Device Corporation’s Quality Department that the welding controls software was developed under a valid software development life cycle (SDLC) because the intended use risk is classified as high. After discussing this issue with the COTS supplier, the project manager learns that the suppliers’ SDLC processes were audited recently by an independent auditing firm. The project manager then contacts the independent auditing firm and purchases a copy of the COTS supplier’s SDLC audit report. The net result is that the Quality Department is convinced that the COTS supplier developed the software under an effective life-cycle model.			

Table C.9 — Example 3 — Validation report

De-vel-op	Im-ple-ment, test and de-ploy	Process	Iterative risk analyses	Validation planning and reporting	Software system
		Validation report (see 5.3.3.5) The project manager completes and gains approval of the validation report.			

Table C.10 — Example 3 — Software release

De-velop	Im-ple-ment/ Test/ De-ploy	Process	Iterative risk analyses	Validation planning and reporting	Software system
		Software release (see 5.3.3.6) The project manager verifies that the software placed under the formal configuration management system matches the software cited in the validation report.			

Table C.11 — Example 3 — Analysis of change

		Process	Iterative risk analyses	Validation planning and reporting	Software system
Main-tain		**Analysis of change (see 5.4)** The project manager verifies that, under the validation plan, the company has a formal change control process that governs any post-validation changes to the welding system.			

Table C.12 — Example 3 — Maintenance validation planning

		Process	Iterative risk analyses	Validation planning and reporting	Software system
Main-tain		**Maintenance validation planning (see 5.4.2)** The project manager thinks ahead to what activities will be appropriate to ensure confidence that the system continues to fulfil its intended use. Given the high risk of the system, the project manager decides that there should be a quarterly calibration and certification that the actual temperature measurement versus the temperature value printed in the lot report is accurate and precise. The project manager includes a section in the validation plan to document this conclusion and issues a request to have a calibration and certification procedure developed and implemented to ensure that this quarterly review is conducted once the system goes into production.			

Table C.13 — Example 3 — Software maintenance

		Process	Iterative risk analyses	Validation planning and reporting	Software system
Main-tain		**Software maintenance (see 5.4.6)** The project manager verifies that, under the validation plan, the company has a periodic review process that ensures that the welding system and process does not vary from its intended use.			

Table C.14 — Example 3 — Retirement of software

		Process	Iterative risk analyses	Validation planning and reporting	Software system
Re-tire		**Retirement of software (see 5.5)** The project manager verifies that under the validation plan, the company has a formal software retirement process that governs retirement of the welding system.			

Toolbox selections

Design, development and configuration tools

— Process requirements definition

— Formal software requirements review

— Identification of risk control measures within the manufacturing and business process

— Process development review

— Traceability matrix (inherent in the requirements specification)

Test tools

— Test planning

— Software system test

— Software configuration control

Deploy tools

— User procedure review

— Internal training for the application

— Installation qualification

— Process validation

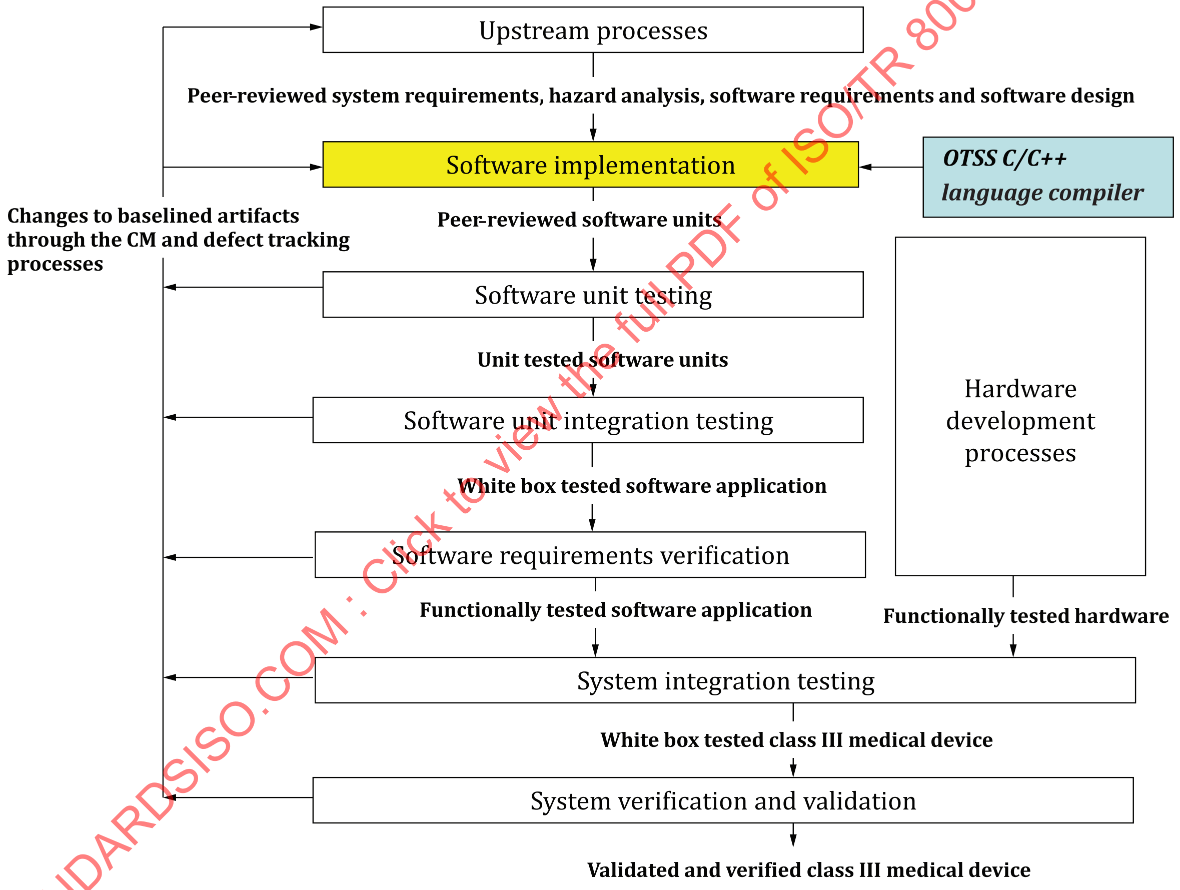
Example 4: C/C++ language compiler

Background

A class C medical device company needs to validate its off-the-shelf software C/C++ language compiler for an embedded system. It has been determined that the compiler is regulated because it produces medical device product software (the software source code and executable software) that is placed in the medical device design records.

Description of the quality system processes

Two quality system processes are pertinent to this case study. The first is the overall quality system process of the implementation of class C medical device software (see Figure C.1). The second is the process to develop the executable software units that implement the software design and meet all the software requirements. These software units include the OTSS C/C++ language compiler (see “Software implementation” in Figure C.1).



Upstream processes
Peer-reviewed system requirements, hazard analysis, software requirements and software design
Software implementation
OTSS C/C++ language compiler
Changes to baselined artifacts through the CM and defect tracking processes
Peer-reviewed software units
Software unit testing
Unit tested software units
Hardware development processes
Software unit integration testing
White box tested software application
Software requirements verification
Functionally tested software application
Functionally tested hardware
System integration testing
White box tested class III medical device
System verification and validation
Validated and verified class III medical device

Figure C.1 — Implementation of class C medical device software

Upstream processes

Upstream of the process to implement the software are the processes to develop the system-level documentation (e.g. requirements, design, hazard analysis) that characterizes the medical device to be developed. The portion of the system implemented in the software is then characterized through processes to develop the software requirements, software design and other software documents or plans. In parallel with software development, additional processes are performed to develop the medical device hardware.

Software implementation process

The formal software language used is the C/C++ software language. An OTSS C/C++ language compiler is used to compile high-level software statements into executable machine code. The output of the software implementation process is the baselined software units, which are peer reviewed by other technical members for completeness and correctness. For a software unit peer review, the software unit should be compiled error-free at the highest compiler level and any compiler warnings should be explained at the peer review.

Downstream testing processes

The software units are tested or verified in several testing processes as follows.

— Software unit testing. The individual software units are tested for logical correctness and for boundary conditions for each unit. This testing might occur on a development system or target system (medical device hardware). Simple software units could forgo this testing when it is determined that a code peer review is adequate for detecting unit logical errors.

— Software unit integration testing. The software units are integrated and tested to ensure that the software design is correctly implemented and that boundary conditions with respect to the design are tested. This testing occurs on the target system.

— Software requirements verification. The complete software application is verified against the complete set of software requirements. This verification is performed on the target system.

— System integration testing. The software and hardware in the medical device are tested to ensure that the system design is correctly implemented and that boundary conditions with respect to the system design are tested.

— System verification and validation. The medical device is verified at the system requirements level and, in addition, is validated for its intended use.

Analysis of process failure risk

The project followed the company's process risk assessment procedure. The overall quality system process of the implementation of class C medical device software (which includes all of the processes described in Figure C.1) is inherently high risk as it generates software that functions within a class C medical device.

The OTSS C/C++ language compiler, as a part of the software implementation process, is assessed to be low risk on the basis of two factors:

— the compiler does not directly cause serious injury or death to a patient, operator or bystander;

— downstream verification is performed on the output (software source code and executable software) of the tool (e.g. software unit testing, software unit integration testing, software requirements verification, system integration testing, system verification and validation).

Intended use definition

The purpose and intent of the OTSS C/C++ language compiler within the software implementation process described above is to author the embedded system source code and to perform the compilation process to generate executable software for a class C medical device.

Software use requirements

a) The tool should cross-compile C and C++ code to work on the reduced instruction set computer (RISC) processor using the selected vendor operating system.

b) The compiler should have a source code debugger.

c) The compiler should be American National Standards Institute (ANSI) C and C++ compliant.

d) The compiler should integrate with the various approved industry standard integrated development environments.

e) The vendor should publish a known bug list that is searchable. The list should be used as a reference to consult as needed.

f) The vendor is required to have a large user base within a regulated industry.

Analysis of software failure risk

A risk analysis of the OTSS C/C++ language compiler reveals that if there is an error, the following events could occur.

— Risk 1. The vendor fails to supply the appropriate business processes, development methods and support capabilities.

  — Mitigation 1. See “Vendor selection process” section below.

— Risk 2. The compiler produces incorrect executable statements.

  — Mitigation 2. See “Validation plan” section below.

— Risk 3. The user, who is not exercising the most rigorous level of error checking, uses the compiler incorrectly.

  — Mitigation 3. Improve training, procedures and work instructions.

Vendor selection process

The project has followed the company’s quality system procedure for selecting and approving vendors, and this information is captured in the project’s design records. This procedure included an on-site assessment reviewing the vendor’s SDLC policies, procedures, tasks and activities. The capabilities of the OTSS C/C++ language compiler offered by the vendor were verified to satisfy the software use requirements defined above.

Validation plan

A downstream validation approach was selected for the OTSS C/C++ language compiler. The vendor selection process has determined that the vendor met all of the documented software use requirements. The compiler has had significant run time at the vendor and will have significant run time during the debug and testing performed on the project. The output of the compiler is subject to the following dynamic testing in downstream processes:

— software unit testing;

— software unit integration testing;

— software requirements verification testing;

— system integration testing;

— system verification and validation.

Validation report

Contents of the validation report are as follows:

— OTSS description

— Software use requirements

  — Hardware requirements

  — Software requirements

  - Patches
- Risk assessment and hazard analysis
- Vendor selection
- Installation activities
- Validation
  - Software use requirements test cases and results
- Known bug list
- Configuration control
  - Training
  - Install location
  - Maintenance
  - Retirement process

Toolbox selection

- Define phase:
  - intended use;
  - validation planning;
  - risk management planning (risk assessment).
- Implement phase:
  - risk control measures;
  - vendor audit.
- Deployment phase:
  - installation qualification;
  - internal training for the application;
  - final acceptance tests.
- Maintain phase:
  - maintenance planning;
  - known issues analysis.

Example 5: Automated software test system

Background

In this example, the manufacturer is a class C medical device manufacturer. The medical devices produced by this manufacturer are controlled by software. The software is architecturally broken up into two major components: the operator console and the real-time embedded control software. The operator console is the primary human interface to the system. The real-time embedded control software is the software that performs the electromechanical control, data acquisition, timing and the like. The operator console software (residing in a PC running an industry standard operating system and database) and the real-time embedded software (residing in an on-board embedded CPU card) are interfaced using a standard Transmission Control Protocol/Internet Protocol (TCP/IP) hardware and protocol interface.

The software manager on the project has decided that it would be valuable to improve the software development and testing process by introducing automated testing of the software. The software manager has decided to initially implement automated software testing of only the operator console software. The automated software testing will take place at both the integration test point and the software system test point.

Determining that the software is regulated

Because the automated test software will be used to perform testing that is required by the manufacturer's software development procedures, and because it will provide evidence of required regression testing at the integration and system test points, the automated testing software was determined to automate part of the development process and therefore was determined to be subject to the validation requirement of ISO 13485.

Defining the process

To better understand the requirements and risks involved in introducing automated software testing of the operator console, the software manager defines the use of the automated test software during the software development process as follows.

During the development of the device software, various modules are scheduled to be integrated into the system software at various times. In addition, modules that have already been integrated into the system will undergo changes because of defect corrections and modifications to requirements. The automated test system is planned to be used for regression testing of the integrated system software and for final testing of a specific module in the system. The software project plan calls for the integration or updates of modules to occur two to three times per week. The automated tests will be run at each of those integration points to ensure that the new functionality works correctly and that previously working functionality has not been adversely affected by the new code that has been added or by changes in the code in a particular build. The automated tests will be run at the software system test level for builds that are candidates for final release to validation and ultimately to customers. The automated testing will also be used in the event that defects are discovered in the final phases of development that need to be corrected to provide a level of regression testing that supplements planned manual testing.

Analysing the risk

The software manager now goes through an analysis process to determine any potential impacts if the use of the automated test software does not go correctly.

The first thing that the software manager needs to evaluate is whether a failure of the automated testing process, a failure of the automated test software or a mistake made by anyone using the automated test software could ultimately lead to a flaw in the medical device that could potentially harm the patient, the operator, a bystander, a service person or the environment.

— The software manager's biggest concern is that the automated software test system will give a false indication that the operator console software under test is working correctly when it actually still has defects.

— If the undetected defects are in a critical area of the software, they could cause a malfunction in the medical device that could create a harm scenario.

— The software manager realizes that such a risk could arise from incorrect management, from use of the automated test software or from a flaw in the automated test software itself.

— The software manager decides that it is extremely important to put boundary conditions around when the automated software test system can be used and what it can be used for to ensure that the software development and test team are not over reliant on the system.

— Individuals who will be involved in configuring, programming and operating the automated test software will need to be trained in their roles.

— The software manager feels that if such factors are controlled, the potential associated risks will be mitigated to an acceptable level.

Defining the software intended use

Having analysed the potential use of the automated test software and the associated risks, the software manager is ready to develop the statement of purpose and intent for the automated software test system. The statement reads as follows.

— The automated test system will be used to test builds of the software at the integration test points during the development process.

— The automated test system will be used to test validation and candidate release builds at the software system test point.

— The automated test system will perform regression testing of the system to ensure that workflows have not been adversely affected by newly introduced software or changed software.

— The general role of the automated test system will be to provide supplementary regression testing to manual testing that will take place.

— For low-complexity, predictable workflows, the automated test system can be used as the final determinant of the correctness of the software, given that the specific protocol has been verified as consistent with equivalent manual testing.

— The automated test system will exercise software that provides safeguards (risk mitigation) for the software system or the medical device as a whole.

Validation planning

The software manager now has a clear understanding of the process that is to be automated, of the specific intended use of the automated test system and of the potential risks involved. The software manager has already determined that certain controls will need to be in place regarding the use of the software and that the automated software testing system, if used in the way that the software manager has prescribed with the appropriate controls, will have an acceptable level of risk associated with its use.

— In this case, the software manager has determined that, when the automated software test system is appropriately used, little or no risk exists that it will contribute to a medical device flaw. The software manager has defined appropriately used as meaning that the software development and test team will not overly rely on the use of the system to determine the correctness of the software. Given the determination of low risk, the software manager has determined that the validation requirements for the system will be on the low end of effort and rigor regarding testing of the software test system.

Validation documentation: Validation report approach

The approach that the software manager has selected is to develop a software validation report for the automated software test system that will include a summary of all of the activities related to gaining the necessary level of confidence in the system.

Critical thinking

The software manager now determines how best to reach the necessary level of confidence, that the system will be used appropriately and that it will not contribute to a serious flaw in the medical device.

He determines that among the most important factors in reaching the necessary level of confidence in the system are the following.

— Strict adherence to appropriate intended use

  — Ensure that all personnel involved in software development and testing clearly understand the boundary conditions and the appropriate intended use of the system.

  — Documentation: Include a section in the validation report that describes the specific intended use and the ways that this information will be communicated through the project's software development plan.

— Due diligence

  — Purchase an industry standard automated software test system from a reputable vendor whose test system is being used for the same level of criticality or more critical applications.

  — Review the intended use of the system with the vendor to determine that the intended use is appropriate.

  — Obtain information about how the vendor validated the software before release to the commercial market. Obtain a statement from the vendor's Quality Department confirming that the commercialized software has been validated by the vendor. This statement will give the confidence that the automated software test system has been adequately tested by the vendor and will establish an initial foundation for the additional activities that the software manager and the software development and test team will perform.

  — Establish a relationship with the vendor to ensure that the software manager and the software development and test team are aware of known issues and defects with the version of test software they will be using.

  — Gain an understanding of the vendor's future plans for software updates to ensure that migration plans to new versions of the software and revalidation activities can be anticipated.

  — Documentation: Include a section in the validation report describing the results of the vendor's due diligence activities, including information on the vendor's validation of the automated software test system, on the method of access to the vendor's defect (bug) list and on the anticipated migration plan to new versions of the software.

— Installation testing

  — Confirm that the computing environment in which the software will be residing meets the vendor's specifications.

  — Establish an initial high-level test protocol with the purpose of ensuring that the software has been installed correctly.

  — Documentation: Include a section in the validation report describing the results of the installation confirmation activities.

— Risk management

  — Ensure that the system will be used only as defined by the software manager in the software purpose and intent.

  — Include specific allowable boundary conditions in the software development plan for the project in which the automated test system will be used.

  - Conduct an analysis to identify the exact coverage areas tested by the system to ensure that manual testing addresses the areas that the automated software test system does not cover.
  - Documentation: Include a section in the validation report describing the risks that were identified in the initial risk analysis and indicate how each of these risks will be mitigated.
- Software use requirements
  - Develop a list of the automated test system functionality that they intend to use. The list, which is developed by the software manager and the software development and test team, is called the "software use requirements" and represents the functionality that will be used.
  - Documentation: Include the "software use requirements" list in a section of the validation report and describe each of the software use requirements.
- Validation of the automated test system
  - Use the "software use requirements" list to determine the necessary level of confidence. The level of confidence can be established by taking three of the initial automated test scripts or protocols and running a side-by-side test against the same protocols run manually. The three initial test scripts or protocols exercise all of the functionality that the team will be using.
  - Documentation: Include a section in the validation report summarizing the results of the side-by-side testing and include evidence of the testing to show that the results were equivalent.
- Training
  - Establish a training program for all system users to ensure that they fully understand how to use the system and are qualified to use it. The software manager believes that training is one of the most important elements needed to ensure the safe and effective use of the automated software test system.
  - Documentation: Include a section in the validation report describing the required training necessary for system users.
- Validation of individual automated test protocols
  - Where the automated test system will be used to test software that is designed to mitigate system, hardware or software risks and hazards, ensure that each protocol has been verified by using side-by-side testing of the automated tests and manual tests.
  - Where the automated test system will be used for final testing of low-complexity, predictable workflows, ensure that each protocol has been verified using side-by-side testing of the automated tests and manual tests.
  - Documentation: Ensure that the software validation records for the medical device include evidence of the side-by-side testing of test scripts or protocols that fit the category.
- Configuration management
  - Ensure that only the appropriate, validated version of the automated test software is installed and being used.
  - As new versions of the automated test software become available from the vendor, control the implementation of such new versions or changes to ensure that the versions or changes are introduced at appropriate times.
  - Ensure that revalidation of the automated test system is considered at each update point and that each revalidation of the system is conducted and documented.
  - Documentation: Include a section in the validation report describing the configuration management plan for the system.

Validation report

As a result of the confidence-building activities, the software manager submits the validation report for final review and approval. The report conveys the thought processes that went into determining the value-added activities to be conducted so that the software manager could conclude that use of the automated software test system result in a scenario in which the associated medical device under development would inadvertently be flawed. The report also contains evidence that all of the activities determined to be important were conducted as planned.

The following are the contents of the validation report:

— process definition;

— risk analysis;

— risk management;

— intended use;

— vendor due diligence;

— training;

— installation testing;

— intended use validation of the automated test system;

— maintenance, revalidation and configuration management.

Validation report review and approval

The software manager routes the validation report to the project manager, the project software quality assurance manager and the software test manager for review and approval.

All reviewers feel that the software manager has clearly thought through the intended use of the system and understands all of the associated risks involved in the system's use. The reviewers feel that all activities that are necessary to reach the level of confidence in the system required to allow the system's use have been performed. The reviewers approve the plan. The system is deemed to be validated and is put into use.

Example 6: A simple spreadsheet

Background

The laboratory analysts at Company ZYX are tired of pulling different specification sheets from their document control system for every product they analyse and then manually calculating the angle number they need to compare against the specification. An instrument in the laboratory is used for receiving inspection. The instrument measures three coordinate locations, which the analysts use to calculate an angle that is compared with the specification. The laboratory has encountered three recent instances in which an analyst has incorrectly calculated the angle (because of "fat fingers," the analyst says) and the analysts wanted to prevent this error from recurring. They decide to create a spreadsheet to perform the angle calculation and to combine the specifications for all 50 products they analyse onto this spreadsheet. They would enter the three coordinate pairs that their instrument measures, select the product name from a pull-down menu and obtain a pass/fail result. The analysts also consider an interface for the instrument to pass the coordinates directly to the spreadsheet, but because of the cost of the interface, this enhancement is delayed until next year.

Definition of the process

The current process contains the following steps.

a) Have the instrument measure the part.

b) Write down the three coordinate pairs.

c) Calculate the angle.

d) Pull the specification for the part from the document control system.

e) Compare the angle value to the specification and determine pass or fail.

f) Put a pass sheet or a fail sheet on the parts and send them into product parts inventory.

The new process will contain the following steps.

a) Acquire the spreadsheet from the document control system.

b) Have the instrument measure the part.

c) Enter the three coordinate pairs into the spreadsheet.

d) Visually check the coordinate pairs entered against the instrument values.

e) Select the part number in the spreadsheet.

f) Select "Calculate result" in the spreadsheet.

g) Visually check that the correct part number was selected.

h) Depending on the result, put a pass sheet or a fail sheet on the parts and send them into product parts inventory.

Definition of the intended use

The analysts define the purpose and intent of the spreadsheet as follows: the spreadsheet will take three entered coordinate pairs, calculate an angle, and then compare this angle to the product specification for the selected product, reporting a pass/fail result.

Risk analysis

The analysts brainstorm the possible hazards related to the spreadsheet. They determine that an incorrect result could mean that parts that did not meet specification could be used in production. For such defective parts to make it to the end user of the medical device, at least two other downstream failures would have to occur, but a slight, if unlikely, risk of harm to the end user remains. Therefore, there is a low risk of producing a product that fails to meet specifications. However, there is a larger risk of increased manufacturing costs, because if the incorrect parts are used in production, they would not be caught until the first subassembly inspection. As a result, the subassembly would have to be scrapped. Moreover, good parts might be thrown away if an incorrect result of fail was received, again increasing the cost of scrap. Therefore, rigor in the form of spreadsheet design, procedural controls, document reviews and testing will be added to address the business concerns.

Validation planning

Because of the low risk of producing out-of-specification product, the level of effort for this validation effort will be low. The analysts decide to combine the spreadsheet requirements and the validation plan into the same document. The analysts also decide to combine the design documentation with high-level test planning. For such documents, the analysts plan reviews by the entire analyst team (four people) as well as by a quality assurance representative. In addition, the analysts plan to consult technical experts to develop a representative set of test data to build confidence that the calculation is functioning as intended. The technical experts will also approve the document.

Risk control measures

The analysts look at each item in the spreadsheet that could introduce error and cause an incorrect result. For each item, the analysts identify how they would mitigate the risk (see Table C.15).

Table C.15 — Example 6 — Risks and mitigations

Risk	Mitigation
Incorrect values could be entered.	Confirm each value pair entered against the instrument through a procedural control. Step d) was added to the new process to do this.
The calculation could be incorrect.	Confirm that the formula is correct and that it provides accurate results as intended.
The wrong product could be selected.	Confirm the part number through a procedural control. Step g) was added to the new process to do this.
The macro to indicate the result could be incorrect.	Confirm that the macro is correct and that it performs as intended.
The specifications in the spreadsheet could be incorrect.	Confirm the spreadsheet specifications against the 50 product specification sheets. Augment the process for specification sheet changes to require an update of the spreadsheet if a specification changes. (This has never occurred but is possible.)
The calculation formula or macro could be changed after validation.	The validated spreadsheet with configuration controls will be put into the document control system and retrieved each time it is needed. The configuration controls will include password protection and locked cells for all non-data-entry cells.

Validation tasks

The formula used is understood and the developer is experienced in spreadsheet macro development. The validation will confirm the following items:

— the calculations;

— the macro;

— cell-locking function (locked cells cannot be changed);

— data-entry checking (values in the allowed range, appropriate product selection, informative error messages).

Because the spreadsheet produces only one result at a time, no stress or performance testing is needed. One test plan and report will be created for all testing. The report will also release the spreadsheet into use and will confirm control of this spreadsheet in the company document control system.

Deployment

Before deployment of the new system, the testing is completed and the manufacturing operators are certified on the operation of the new vision system.

Tools from toolbox

— Requirements definition (documented in the validation plan)

— Process failure and risk analysis (documented in the validation plan)

— Intended use (documented in the validation plan)

— Validation plan

— Test planning

— Operator certification

— Maintenance planning (which calls for regression analysis)

Maintenance

Maintenance will be required on the spreadsheet every time a product specification changes or a new product is added. A maintenance test plan will be developed with a representative subset of the full validation test cases to ensure that new items do not break the spreadsheet. The maintenance plan will call for regression analysis to see if additional test cases need to be added to this subset of test cases specific to the change being made. This plan will also describe how to update the spreadsheet (e.g. unlock the cells, change, relock).

Example 7: A (not so) simple spreadsheet

Description of software

A software development team has used a Microsoft Excel[1)] spreadsheet as a development aid. The spreadsheet will record device message translations used in a class C or D device. The original release of the device was written in US English. Subsequent releases will support seven languages. The spreadsheet consists of seven columns. The leftmost column is the English-language device message for every message in the device. Each of the remaining columns represents one of the international languages to be supported and each row within a column represents the translation from English to the international language for the particular English-language message in the leftmost column of that row.

Intended use

The spreadsheet satisfies transient needs to

— visually organize the English-language messages and their translations,

— create a spreadsheet that can be sent to local representatives for the purpose of collecting translated messages either directly into the spreadsheet or in handwritten form on a hard copy of the spreadsheet, and

— provide a transient data storage tool for the translated messages.

Once the translations are collected and translated into device software, there is no need to keep or maintain the spreadsheet.

No computed cells or macros are part of this spreadsheet.

Determine if the software is in scope

Excel is used simply to format the information for circulation and collection of foreign-language translations of the device messages. At first glance, the spreadsheet appears to be such a simple application of Excel that one is tempted to decide that it does not need validation.

In 5.2, the following question is asked: "Could the failure or latent flaws of the software adversely affect the safety of medical devices or quality of medical devices?"

The answer to this question is clearly "yes." If the software or spreadsheet fails in such a way that it corrupts the message translations that are stored there, the failure could affect the safety of the device. Although the team feels that the likelihood of failure for this "simple application" is low, the likelihood is still within the scope of ISO 13485 validation requirements.

Risk assessment

If device messages are not translated properly, user confusion or misinterpretation of messages could result. Hence, the potential exists for indirect harm to a patient using the device. Failure of the software would be detectable and there are numerous opportunities for cross-checks in the device development and validation process to detect and correct any failures of the software.

Anticipated failure modes that could adversely affect the device software are as follows:

— corruption of original English-language messages to be translated by loss of the entry file, by loss of individual messages, by misordering of messages, thereby leading to loss of context, or by corruption of individual messages by random loss, substitution or transposition of characters;

— corruption of individual translated messages as prepared and collected from regional offices. Corruption might be due to loss of the entry file, to loss of individual messages, to misordering of messages, thereby leading to loss of context or to corruption of individual messages by random

1) Microsoft Excel is an example of a suitable product available commercially. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO of this product.

loss, substitution or transposition of characters. Additional potential exists for corruption of any language requiring non-English fonts if the fonts are not properly installed in Excel;

— corruption of the collected results spreadsheet, which shows the accumulation of results for each translation. Corruption might be due to loss of the entry file, to loss of individual messages, to misordering of messages, thereby leading to loss of context or to corruption of individual messages by random loss, substitution or transposition of characters. In addition to misordering of rows in the spreadsheet, misordering of columns could also occur. Columns that do not display their translated messages in the native fonts and character sets will be misinterpreted by the software engineers as they translate the messages into code.

Validation planning

The software development engineers recognize the potential risk to a patient if the messages for the new device are wrong. The severity of a failure of the software could be high. Something needs to be done to build confidence that the messages organized in the spreadsheet are the correct translations.

However, Excel is being used only to organize the information. It seems unlikely that any amount of testing of Excel will uncover any defects that will corrupt the messages. As the engineers consider this problem, they complain that human error is far more likely to lead to a mistake than a simple application of Excel.

In thinking about human error, the engineers realize that no well-defined processes exist to collect the translations or to verify that no human error has crept into the process.

The engineers create a written procedure for collecting and verifying translated messages. They then consider what risks might exist for their process to break down, how software (i.e. Excel spreadsheet) failure could contribute to that breakdown, and finally, what can be done to validate the process, including the spreadsheet.

Risk control measures

After better defining the translation-gathering process, the engineers identify risk control measures to protect the process from embedding errors into the message translations.

The risk control measures that protect the process of translation collection will also protect from failure of the software to meet its intended use.

— When sourced from regional offices, translations should be provided in either paper (hard-copy) format or in electronic format with an accompanying hard copy. If an electronic version is provided by the regional office, the data in that spreadsheet will be verified (and documented) against the hard copy when it is transferred to the master translation spreadsheet. This verification will protect from any misinterpretation of results caused by corruption of the spreadsheet during transmission or differences in font capabilities between the computer sourcing the translation and the computer receiving the translations.

— Once all translations are collected and placed in the master spreadsheet, a hard copy spreadsheet should be sent to each regional office for review and approval. This regional approval will protect from any misinterpretation of results caused by corruption of the spreadsheet during transmission or differences in font capabilities between the computer sourcing the translation and the computer receiving the translations.

— Once the master spreadsheet is accepted by all regional offices, development approvers and quality assurance approvers, the hard copy of the master spreadsheet should be the input to the software development process for the device. Furthermore, the hard copy of the master spreadsheet should be the source of any expected results from verification testing of translations in the device software.

Validation tasks

In addition to those risk control measures, other verification and validation tasks should be completed to ensure that the software adequately fulfils its transient intended use. These tasks are as follows.

— For each translation collected from regional offices, a hard copy of the updated master spreadsheet should be verified line by line against the hard copy of the individual translation spreadsheet's hard copy. It is mandatory to verify hard copy against hard copy to rule out any mistranslations caused by font differences between computer platforms or printers.

— A version control process should be documented in detail. The process should specifically account for the following:

  — changes in the message requirements (i.e. English language) as the functionality of the device evolves during development;

  — changes in the master document as translations are provided and as the updated master spreadsheet is reviewed and modified by the regional offices.

— Although the spreadsheet is very simple, some very real version control risks are associated with its use.

— The configuration for the spreadsheet should include the version number of the spreadsheet itself, the version of Excel used, the computer platform configuration and the printer configuration for the printer used to create the hard copy of the spreadsheet. The full configuration is important because font differences can exist in different installations of Windows or Office and in different versions of printer firmware. The only way to be sure the translations do not change unintentionally is to use the same configuration when using the spreadsheet.

— The configuration of the spreadsheet (i.e. operating environment and versioning) needs to be controlled to prevent chaotic, uncoordinated change. A single person is assigned responsibility for deciding when to make changes to the configuration and when to document the change history.

— The version of each spreadsheet should be visible in its hard-copy version.

— Translation tables in the device software should indicate which version of the hard-copy master spreadsheet was used as input to the translated message software.

— The individual translation verification task should include the following.

  — The English-language message should be verified line by line by comparing the master version and translation version of the spreadsheet. This comparison protects against any corruption (e.g. damaged or missing messages) of the spreadsheet file that might have occurred when the file was transmitted to regional offices and when the file was returned by the regional offices.

  — After the translations are inserted into the master spreadsheet (either manually or by using the cut/paste functionality of Excel), a line-by-line comparison of the hard-copy output of the revised master spreadsheet should be verified against a hard copy of the translation spreadsheet.

  — When the device software is tested for implementation of messages, the test procedures should use a hard copy of the latest version of the master spreadsheet (and should reference the version number) for making comparisons of implemented messages to intended messages.

All of these validation tasks should be documented and collected as objective evidence of the validation of the process and of the Excel spreadsheet.

This validation approach results in 100 % verification of the outputs versus the inputs of the software. No further testing of the spreadsheet is planned. Despite the lack of traditional testing, the engineers feel confident in their process and believe that their validation rationale has been a valuable exercise. The engineers reason that any failure of the software would be detected and they have a recovery path through the hard copies that are collected and recorded at the appropriate points in the process. The hard copies and documented line-by-line verifications provide documented evidence of the activity.

Maintenance

The spreadsheet is intended to fulfil a transient need. It is to be retired once the translated messages have been embedded in code. No maintenance plans are created.

Discussion

The intended use and initial risk analysis of the spreadsheet were critical to the determination that the spreadsheet required further validation attention. Under other intended use circumstances, the very same spreadsheet might well have led to a conclusion that the spreadsheet was of low risk and certainly of low complexity. Had the intended use been simply for tracking the progress of the gathering of translations (i.e. the translations on the spreadsheet would not have been used in the design to implementation activities), then the determination might have been that virtually no risk existed to the device's integrity and that, in fact, the spreadsheet was a business management tool and did not even fall under the scope of the regulation.

The "process" that this software was "automating" was part of the process of data collection, formatting and storage of message translations for a device. The example is interesting from several perspectives.

— The validation required little, if any, software testing to validate the use of the software. It is important to note that the software (Excel) and the spreadsheet were validated for this specific use but were not validated generically for any use. The team felt that testing was unlikely to uncover any defects in the software but that there was a vulnerability to the device if the software did fail in some unpredictable manner.

— The validation consisted of 100 % verification of the outputs of the spreadsheet. The hard-copy versions were relied on as the "gold standard." Once the hard copies were approved and used in the design history file, any subsequent failure of the software was inconsequential. Any failure of the software before approval would be caught by the review and approval process.

— The "process" was modified to make it immune to any failure of the spreadsheet software.

— The engineers believed the likelihood of human failure to be much higher than the likelihood of software failure in this application. Users could make typographical errors, could use the wrong version of the spreadsheet or could make similar errors. In this case, the "software validation" also made the process more immune to human error.

— The example makes a strong point for the importance of configuration management, even for routine office productivity tools.

NOTE This example was based on a real case that was not so cleanly handled. In the real situation, human errors occurred with versioning of the spreadsheets. Unexpectedly, issues related to versions of fonts that were linked to different installations of Excel on different PCs gave different hard-copy results. (Printer fonts also became problematic on different printers.) The seemingly simple spreadsheet, one very nearly dismissed as not needing validation, actually became problematic in its corruption of message translations.

Example 8: Parametric sterilizer

Mary has been tasked with leading the validation effort for a new automated sterilizer system that will be custom developed for use by her company, Always-Safe Medical Device Company.

Defining the process

Mary begins by first defining and documenting what she knows about the 100 % ethylene oxide (ETO) sterilization process that is being introduced into her plant.

— Medical devices are manually put into the sterilizer. This process includes sterilization-cycle parameter evaluation to support parametric release.

— The automated sterilizer system software controls sterilization-cycle activities.

— Medical devices are manually removed after the cycle is complete and are transferred to a degassing chamber.

Analysing the process risk

Mary is very concerned about the risk posed by this process. Failure of this process could have severe consequences, including the following:

a) improper sterilization of medical devices. This failure could result in serious injury or death attributable to infection from the use of a nonsterile product;

b) loss of device history information and product traceability;

c) release of toxic chemicals into the manufacturing facility or the environment. This failure could result in the serious injury or death of sterilizer operators or individuals in the local neighbourhood.

Mary therefore considers what risk control measures should be put in place and verified to mitigate these risks. Mary believes risk can be controlled through the use of parametric sterilization techniques to ensure that the right amount of gas is used for the proper time period at the correct temperature and correct relative humidity. Furthermore, manually checking data from the sterilizer for proper parametric values will independently confirm that the sterilization is adequate. Finally, she believed fail-safe shutdowns and containment structures are needed to control chemical leaks into the facility.

With these risk controls in place, multiple simultaneous system failures would have to occur to result in a nonsterile device. However, because of the impact if such failures occur, Mary determines that the residual process risk is high. Therefore, a rigorous validation is appropriate.

Defining the software purpose and intent

Mary wants to have a detailed understanding of how the software in this system will be used. First, she considers what the software is supposed to do. In this case, the software controls the process of sterilizing medical devices using a 100 % ETO sterilizing vessel, including the recording of information for inclusion in the device history record and the analysis of sterilization values to support parametric release. The new sterilizer was purchased because it can accommodate larger batches than can the current system; larger batches are needed to meet current product demand. The sterilization operators will be using the system, along with the quality assurance team, to determine the acceptability for release of medical devices. Mary understands that this effort will be carried out through real-time control and monitoring of the sterilization vessel during sterilization cycles and storage of information in a database. Mary is pleased to learn that the system will be physically located in the site sterilization facility and, further, that the system will generally be shut down one day a week to allow for any necessary maintenance.

Mary determines that the software will automate all aspects of the sterilization cycle, from the point of manually placing the devices into the vessel to the point of manually removing the devices from the vessel.

Mary documents the software's purpose and intent as follows.

— The sterilization software will control and monitor the sterilization process and will evaluate sterilization-cycle parameters for parametric release.

Validation planning

Now that Mary understands what the software is intended to do, she is ready to develop the validation plan at a high level. She knows that she will need to add more detail later but wants to begin validation planning now so that she can identify software failure risks in an informed manner and use the identified risks to complete her planning.

Because of the high residual process risk that she identified earlier, Mary believes that she needs to provide detail and formality in the validation effort. She expects to use a high level of rigor and detail in the documentation and to have most documents as stand-alone documents rather than combining them, as is often done for smaller efforts. Because of the high risk associated with the system, she decides to treat development with the same level of rigor that she would use for developing medical device software. Consequently, she decides to follow IEC 62304:2006/AMD1:2015 as a life-cycle control methodology. For guidance on software risk management, she refers to IEC/TR 80002-1. Furthermore, to be sure that all the potential sources of harm are considered, Mary decides to apply software fault tree analysis to the development effort. She also decides to formally define and document user business process requirements and software requirements. Any functionality of special concern will be specifically identified. Mary also schedules a formal software requirements review. Approval will be required by the quality assurance team, the sterilization engineer and the manager of sterilization. Because of the criticality and risk of this system, the final approval of the validation report will include members of senior management.

Defining software requirements

Mary now writes the software requirements definition. She decides that the software requirements should deal with alarms, error handling and messages, confirmation of parametric settings, interface to the device history record system, sensor control and monitoring, motion control and monitoring.

Establishing confidence and control over the software

Using Always-Safe's internal development control procedures as a driver, Mary uses internal controls throughout the development life cycle. Because everything is done internally, no vendor activities need to occur.

Defining software boundaries with other systems

Mary then considers what other systems the new sterilizer will need to interface with. She determines that the only interface will be with Always-Safe's existing database system, which will store data generated during sterilization cycles.

Analysing software failure risks

Although Mary has already determined that the business process being automated is of high risk, she still needs to analyse the risk of a software failure. Using this document as a reference, Mary selects a quantitative risk model for this activity. She ranks the new system as follows.

— Risk of "severity" is high (10) because the failure of the system could lead to death or serious injury.

— Risk of "likelihood" is also high (10) because a software failure itself could lead to harm, because the software is making the determination about the acceptability of the sterilization.

She calculates a risk score of 20, which translates to a high-risk classification. High-risk classification means a rigorous validation methodology should be applied. The methodology being followed is as rigorous and comprehensive as if the sterilizer were itself a medical device.

Because of mitigations, the residual risk of this automated system is as low as reasonably possible. Because of the severity of harm that comes from the system, sterilization is inherently a high-risk process. Additional activities related to the risk, drawn from IEC/TR 80002-1, are also performed.

Finishing the validation plan

Because Mary has now completed definition of the software requirements, has decided on the implementation approach and has analysed the software risk, she has enough information to complete the detailed validation plan.

In writing the first draft of her validation plan, Mary has already decided that a rigorous approach to risk management should be taken. She has already planned to treat the validation effort in a highly formal way.

Accordingly, she delineates the risk management tools (identified in IEC/TR 80002-1) that she plans to use as follows.

— Risk management tools:

  — software fault tree analysis;

  — risk management plan;

  — identification of risk control measures within the manufacturing or business process;

  — analysis of software failure (risk analysis).

Mary then considers how she will gain confidence in the software during software design, development and configuration phases. She has already decided to follow IEC 62304:2006/AMD1:2015 for life-cycle controls. She now identifies other specific tools that she will use to ensure that the software is developed properly during the design, development and configuration phases.

— Design, develop and configuration tools:

  — IEC 62304:2006/AMD1:2015 Architecture documentation and review;

  — design specification;

  — software detailed design and review;

  — software coding standards;

  — traceability matrix;

  — identification of risk control measures within the software system design;

  — code review and code verification;

  — development and design reviews.

Mary has no doubt that she will need to test this new system extensively. She decides first that a formal test planning activity will be needed along with the usual unit testing, integration testing and interface testing activities. However, because this system will be releasing finished devices in real-time, she decides that she needs to push the limits of the system through stress testing, performance testing and more extensive combination of inputs testing to mimic as many operating conditions as possible.

— Test tools

  — test planning;

  — unit tests;

  — integration tests;

  - interface tests;
  - regression tests (as necessary);
  - software system test;
  - robustness (stress) tests;
  - combination of inputs tests;
  - performance tests.

Finally, knowing that the system is not complete until it is fully implemented in the production environment, Mary turns her attention to the validation activities that she would like to see during the deployment stage. She wants to be sure that the system is adequately documented and that users are well-trained in its correct use. She also wants to be sure that the system is actually installed as intended. So Mary's validation plan for the deployment phase now includes the following items:

- Deploy tools:
  - use procedure review;
  - internal training;
  - installation qualification;
  - operational and performance qualification;
  - operator certification.

Planning for maintenance

Mary is concerned about maintenance of the software because of the high residual risk. She plans several maintenance activities to ensure software quality once the system is deployed, including evaluation of the effectiveness of user training, system monitoring techniques, correctness checking of system outputs and defect reporting. She also confirms that calibration and other hardware maintenance activities are occurring in addition to the software maintenance activities.

Retirement activities

Mary struggled over retiring the previous system because the data generated by that system needed to be archived for device history record purposes and the old format was not compatible with the new format. The new system uses a universal data format to allow flexibility upon its retirement in migrating the remaining data to a new system.

Example 9: Nonconforming material reporting system — Total system upgrade

Advanced Medical Specialities Corporation is upgrading its nonconforming material reporting system (NCMRS) software, a commercial software package. Advanced Medical chose not to upgrade at the last major release, so the system is now operating two major releases behind. (Advanced Medical is currently running version 2 and the latest release is version 4.) To maintain the current software maintenance agreement, Advanced Medical needs to upgrade. Version 4 of the NCMRS-Pro software has significantly changed over previous versions. Among other things, the product has been re-platformed from a typical client-server application to a web-based application. The new software also includes significant new features and functions. Frank, the business process owner and project manager at Advanced Medical, has no new requirements over his existing software and process, but he does wish to take advantage of the new software features.

Frank consults with the regulatory team and decides that the current interface between the ERP system and the NCMRS can remain intact without modification. Frank recognizes, however, that the new version is capable of writing data back to the ERP system and that this expanded interface should be thoroughly challenged during the validation. Frank and his colleagues, the manufacturing quality engineer and the regulatory team, begin the exercise of determining the scope of the validation effort. This group is referred to as the “team” throughout the rest of this example.

Defining the process

Frank begins by analysing his current manual process to determine what elements of the workflow will be automated by the new software. The new software will change the following features:

a) recognition of potential nonconforming materials or products (out of scope);

b) input of information related to the material and the circumstances surrounding its discovery (in scope);

c) routing of information to allow for proper identification, evaluation, investigation and disposition of the material (in scope);

d) distribution of information to important stakeholders and to other computer systems that are required for proper handling of financial, purchasing, planning and scheduling transactions (in scope);

e) physical disposition of the material, although pertinent data about the disposition will be recorded in the system (out of scope).

Analysing the process risk

Frank is aware that this process and the supporting software carry a risk. Failure of the process could have serious consequences, including the following:

— inadvertent release of nonconforming materials onto the manufacturing floor;

— inadvertent release of nonconforming product into commercial distribution;

— increased cost or manufacturing attributable to scrap, rework and the like.

Frank and the regulatory team consider what risk control measures are in place to mitigate these risks, including the following:

— procedural controls to detect, segregate, control and correct nonconforming materials;

— management and quality review of statistical process control data and other measures to identify developing trends that can signal when processes are not in proper control;

— ongoing training of operators to ensure compliance with procedures;

— financial reports to help identify material use that would suggest uncharacteristic issues with manufacturing processes.

With those risk controls in place, multiple simultaneous system failures should occur to result in a failure to appropriately control nonconforming materials or products. However, because of the potential quality, regulatory and financial impacts of such failures, Frank determines that the residual process risk warrants rigorous confidence-building activities to help ensure that the software is operating correctly and that it meets the intended use.

Defining the software purpose and intent

Frank wants to have a detailed understanding of how the software upgrade will affect his users and the organization. Frank concludes that the software is essentially an automated issue tracking and management tool. Manufacturing personnel who work with standard tools, equipment and other instruments are responsible for recognizing and isolating potential nonconforming materials and products. Once an issue is recognized, details about the situation are entered into the software. The software then manages the workflow, assignments and notifications to resolve the issue and documents the various activities that are necessary to deal with the disposition of the materials and products. The software upgrade should streamline the process, thus making it more efficient, and it should provide the quality assurance team with more powerful tools to analyse and trend data and give the team greater visibility over quality issues. Frank understands that the process changes necessitated by the upgrade are primarily to the workflow and the distribution of information. The software itself makes no final decisions, nor does it independently determine any outcomes, but the software does hold and document the decisions made by humans interacting with the system.

Frank determines that the software will automate the workflow aspects of the non-conformance processing. The following statement of the software’s purpose and intent was composed by the regulatory team.

— The NCMRS software is intended to support the processing of nonconforming materials and products. The system is used to document the process steps, as defined by SOP, and records the process steps performed, the time they were performed, the personnel who performed them and the outcome of each step. The system makes data readily available for quality monitoring and improvement activities.

Defining software boundaries with other systems

The NCMRS software has two interfaces, including one primary interface with the ERP system and one secondary interface with the company’s human resource (HR) system. The primary interface is designed to be a twice-daily scheduled batch process to update the system with data on finished goods, in-process goods, bill of material and bill of operations. The interface will also supply nonconforming material report (NCMR) data back to the ERP with data about quality holds, material disposition and other transactional information. The secondary interface is unidirectional from the HR system and is intended to update NCMR employee data for scheduling and assignment purposes.

Initial validation planning

Frank has gained additional confidence that the NCMR process and software are adequately understood and properly documented. He is now ready to develop a high-level validation plan. Additional details will be developed during the planning process.

The regulatory team identifies the documents that will add the greatest value and will adequately specify what the software is expected to do. The documents will generally be referred to as “Requirements,” but will not be a typical set of user requirements per se. Instead, the documents are going to be a series of detailed descriptions of how the software is expected to operate. As such, the analysis of automated and manual testing will be more qualitative in terms of its review and outcomes. It will look holistically at the output and whether the system is performing as intended, rather than looking at individual tests to determine if specific user requirements have been met.

The document set will include the following.

a) Workflow and business rule documentation. This area of the software is configurable, so the team will prepare the desired set of configurations it intends to make and will develop detailed process flows and logic diagrams that describe the operation.

b) Interface documentation. These documents will describe what data elements move from the ERP and HR systems to the NCMRS, what data elements move from NCMRS to the ERP system and when such elements move.

c) Data migration documentation. This set of documents will describe what historical data will be moved to the upgraded system.

The validation plan will include the outcome of the review and approval of each of these documents. Approval will be required by the quality assurance team, the manufacturing engineering department and the information systems group. Because of the system's criticality and risk, all members of senior management should give final approval to the validation report.

Defining software use requirements

Frank and members of the team go about the business of assembling the document sets described above, referring to the system documentation supplied by the vendor and the previous documentation of the existing interfaces.

Establishing confidence and control over the software

Frank has had a positive experience with this software and the vendor. Frank now identifies five main efforts that the team will use to establish confidence in the software.

a) The vendor has an approved status with the company in accordance with Advanced Medical's internal policies and procedures. Previous audits have revealed that the vendor has an adequate quality system and SDLC. The vendor produces commercially available software that has an established history in the regulated industry for uses similar to Advanced Medical's intended use. The vendor will be periodically audited to maintain the approved status.

b) Advanced Medical will use a vendor-supplied automated testing tool to verify that the software has been installed correctly and is functioning within the boundaries of the test suite. This tool can process more than 8 000 various transactions in several hours. The tool does not, however, test certain configuration options that the company plans to include.

c) The team will produce an adjunct test plan that includes parallel processing of a statistically significant sampling of actual paper-based non-conformance reports. Outputs will be reviewed to ensure accuracy, data integrity and compliance with procedure.

d) The team will verify data conversion and migration of existing system records by using a sampling technique that ensures that historical records retain their integrity. Record counts will be used to verify 100 % conversion.

e) Data interfaces will be verified using a sampling technique to measure completeness and accuracy of the data transfers.

Analysing software failure risks

Frank uses this document as a reference to determine the validation rigor that is going to be required. Software failures could result in loss, corruption or mishandling of electronic records. Mitigation of the risks is controlled by the vendor's internal quality systems, by the installation qualification of the software (automated test tool) and by adjunct use case testing and verification. Because of the downstream process controls, the residual risk of this system is deemed to be as low as reasonably practicable.

Final validation planning

This determination implies that a fairly rigorous validation methodology will be applied. The methodology being followed ensures, to a reasonable extent, that the software will perform as intended. The team members conclude that they have adequately defined the requirements of the system, that they have decided on the implementation approach, that they have analysed the software risk and that they have obtained enough information to proceed with a detailed validation plan.

The bulk of testing to be performed will be accomplished using an automated test suite, which the team has reviewed and has determined to be valid for this intended use. Additional adjunct use case testing will be conducted using actual business cases originating from the manufacturing floor. The purpose of these tests is a) to verify that the process works as intended, b) to accelerate user acceptance and training and c) to verify that configuration changes have not adversely affected the software. The adjunct testing is not intended to replace the vendor's internal system testing, which has previously been verified by an audit. Successful completion of the automated testing will establish that the software is correctly installed and is functionally acceptable.

The team selects the following tools from this document to conduct the remaining installation, configuration, testing, verification and validation efforts.

— Design, development, and configuration tools:
  - — architecture documentation and review;
  - — identification of risk control measures within the software system design;
  - — configuration design reviews;
  - — review of vendor's "known issues" list;
  - — review of vendor's base system validation documentation;
  - — review of "out-of-the-box" software workflow process diagrams;
  - — review of the "out-of-the-box" standard reports library;
  - — gap analysis of configuration changes made to standard workflows and business rules.

— Test tools:
  - — test planning;
  - — description and outcomes of the vendor-supplied automated test tool for installation verification and qualification;
  - — installation and performance testing (part of the automated test suite);
  - — use case testing to cover configuration changes using actual non-conformance records as inputs rather than artificially constructed test cases;
  - — sampling plan to verify migrated data;
  - — system checks to verify operational interfaces.

— Deploy tools:
  - — use procedure review;
  - — internal training;
  - — operator certification.

Planning for maintenance

Frank plans to use several maintenance activities to ensure ongoing software quality once the system is deployed, including evaluation of the effectiveness of user training, system monitoring techniques, periodic auditing of system outputs and defect reporting, both internally and to the vendor. Frank has established a point of contact with the vendor so that notifications of bugs, maintenance releases and other communications come to the attention of the proper staff people responsible for maintaining the software at Advanced Medical.

Retirement activities

Frank plans to keep the current system available once cut-over has taken place as an opportunity to compare throughput and outcomes from which performance metrics can be compiled. After the new upgrade has been successfully operational for 6 months, Frank will completely decommission the previous system.