



INTUITIONLABS / RESEARCH & PRACTICAL GUIDANCE

IEC 62304 Guide: Medical Device Software Safety Standard

Build practical AI for pharma and biotech with IntuitionLabs. We help life-science teams turn complex information and workflows into useful software, governed knowledge systems and AI tools.

Published by IntuitionLabs · 2026-02-01 · iec 62304 · medical device software · software lifecycle processes · samd · iso 14971 · fda software guidance · soup software · medical device compliance

Original article: <https://intuitionlabs.ai/articles/iec-62304-medical-device-software-guide>



In this article

1. Executive Summary
 2. Introduction and Background
 3. Scope and Purpose of IEC 62304
 4. Safety Classification (IEC 62304:2006/2015)
 5. Key Processes of IEC 62304
 6. Integration with Quality and Risk Standards
 7. Regulatory Landscape and Global Perspectives
 8. Development Life Cycle Processes (Clause 5)
 9. Support Processes (Clauses 6–9)
 10. Implementation and Best Practices
 11. Validation Boundary
 12. Process Evidence and Analysis
 13. Discussion: Implications and Future Directions
 14. Conclusion
 15. References
-

Executive Summary

Medical device software (classified as either **Software as a Medical Device** [SaMD] or *Software in a Medical Device* [SiMD]) plays an increasingly critical role in modern healthcare, driving innovation and improving patient outcomes. IEC 62304 applies to the development and maintenance of medical device software (**FDA recognition record**). FDA recognizes IEC 62304 Edition 1.1 as a consensus standard for medical device software (**FDA recognition record**).

This comprehensive report examines IEC 62304 in depth, including its **scope, safety classification scheme, and core processes**. It situates the standard within the broader regulatory environment (e.g. **EU MDR**, FDA regulations, ISO 13485 QMS, ISO 14971 risk management) and compares it to related standards. We analyze real-world cases to illustrate the stakes of software safety, present expert perspectives on best practices, and discuss implementation challenges (including managing legacy software or “SOUP”, *Software of Unknown Provenance* (www.softwarecpr.com), and adopting agile methods within a regulated framework (www.riskmanager.net) (www.riskmanager.net)). The report also discusses the IEC drafting team’s proposal to extend IEC 62304 to health software, including non-medical software (**IEC design specification**).

Key findings include:

- **Fundamental Role of IEC 62304:** It is the *de facto* global standard for medical device software life cycle processes, emphasizing traceable requirements, risk-managed design, software verification, and controlled maintenance. Device validation and final release are outside IEC 62304’s scope. It explicitly links to quality management (ISO 13485) and risk management (ISO 14971) (quickbirdmedical.com) (www.qualityfwd.com).

- **Risk-Based Classification:** Software is classified as Class A, B, or C based on potential harm from failure (www.iss-ag.ch). Class A (lowest risk) means failures cause no injury or only minimal, acceptable risk; Class B indicates non-serious injury; Class C indicates the potential for serious injury or death (www.iss-ag.ch). The depth of development controls (e.g. review detail, testing extent) scales with class – Class C requires the most rigorous processes. (Table 1 summarizes safety classes and examples.)
- **Comprehensive Lifecycle Processes:** IEC 62304 mandates a full software life cycle: detailed planning, requirements definition, architecture/design, coding, integration, testing, release, plus separate maintenance, risk management, configuration management, and problem-resolution processes (quickbirdmedical.com) (visuresolutions.com). These processes are designed to ensure software safety and to create a complete audit trail (e.g., traceability of requirements to design to tests) for regulatory review (www.riskmanager.net) (www.qualityfwd.com).
- **Regulatory Alignment:** Under the MDR and IVDR, presumption of conformity depends on publication of a harmonised standard's reference in the Official Journal and applies to the requirements it covers (European Commission). IEC 62304 Edition 1.1 applies to the development and maintenance of medical device software (FDA recognition record). It also dovetails with other standards (e.g., IEC 60601-1-4 for programmable systems safety, IEC 62366 for usability) to address broader safety aspects of software medical devices (quickbirdmedical.com) (bluegoatcyber.com).
- **Implementation Considerations:** Effective compliance requires strong quality management (ISO 13485) and integration of risk analysis (ISO 14971) (quickbirdmedical.com) (www.qualityfwd.com). Experts note that IEC 62304 “cannot be done with informal or ad hoc processes” (www.qualityfwd.com); rather, organizations must establish or extend their QMS to cover software specifics (SOPs, templates, eQMS tools) (www.qualityfwd.com) (www.qualityfwd.com). Agile development is compatible but requires documented iteration and traceability built into sprints (www.riskmanager.net) (www.riskmanager.net). Integrating third-party code (SOUP) demands including its failure modes in hazard analyses (www.softwarecpr.com) (www.softwarecpr.com).
- **Emerging Trends:** IEC’s design specification proposes two software process rigor levels for a future edition (IEC design specification).

In sum, **IEC 62304 is the cornerstone** for medical device software safety. Its rigorous, evidence-based approach addresses known failure modes and guides manufacturers to systematically manage risks. IEC's second-edition design specification provides guidance and mapping for drafting the second edition of IEC 62304. As software-driven healthcare continues to grow, understanding and applying the principles of IEC 62304 will remain essential.

Introduction and Background

The Digital Transformation of Medical Devices

In recent decades, medical devices have undergone a profound transformation as software becomes embedded in nearly every modality of healthcare. Modern devices – from implantable pacemakers and infusion pumps to diagnostic imaging systems and smartphone apps – rely on sophisticated software to

function. FDA says use of Software as a Medical Device is continuing to increase ([FDA](#)). Globally, software-driven devices promise improved diagnostics, personalized therapies, and remote monitoring capabilities. However, they also introduce new hazards. Unlike mechanical failures, software defects can produce unpredictable and potentially catastrophic outcomes if not managed methodically.

Software safety depends on disciplined development, verification, risk management, and change control throughout a device's life cycle.

Regulators responded by updating frameworks. Early device directives and standards (e.g. IEC 60601 electrical safety) did not specifically address software life cycle. In the late 1990s, voluntary guidance like AAMI's Software Validation series and FDA white papers emerged. Ultimately, in 2006 the International Electrotechnical Commission (IEC) published **IEC 62304**, an international standard dedicated to *medical device software life cycle processes*. IEC 62304 provided the first comprehensive, consensus-based requirements for developing and maintaining safe medical software. It built on fundamental quality principles (ISO 13485 for QMS, ISO 14971 for risk) while filling the software-specific gap.

IEC 62304:2006 was amended in 2015. The [FDA recognition record](#) identifies the consolidated Edition 1.1 (2015) and limits its scope to development and maintenance of medical device software.

This guide examines the standard's scope, processes, classification, supporting standards, implementation, and emerging software practices.

Scope and Purpose of IEC 62304

IEC 62304 establishes a **framework for the safe development and maintenance of medical device software**. Its official title is "*Medical device software – Software life cycle processes*". The standard covers any software that is either a medical device in itself or part of a medical device ([quickbirdmedical.com](#)). In broad terms, it applies to:

- **Standalone medical software (SaMD):** Software running on general-purpose hardware, mobile devices, or in the cloud, whose intended purpose is medically therapeutic, diagnostic, or health-related ([quickbirdmedical.com](#)) ([www.fda.gov](#)). Examples include clinical decision-support apps or chronic monitoring software.
- **Embedded/device software (SiMD):** Software integrated into a physical medical product (e.g. CT scanners, infusion pumps, pacemakers) ([quickbirdmedical.com](#)).
- **Firmware systems:** Software within a device's embedded control systems, often with real-time constraints.

IEC 62304 does not cover validation and final release of the medical device, even when the medical device consists entirely of software ([IEC](#)).

A key feature of IEC 62304 is its risk-based approach: its software life cycle activities are organized around software safety classification.

In addition to lifecycle activities, IEC 62304 explicitly references related standards. For instance, Clause 4 (*General Requirements*) mandates that the manufacturer have a quality management system (e.g. ISO 13485) and perform risk management per ISO 14971 (quickbirdmedical.com). Consequently, IEC 62304 is designed to be used *with* these complementary standards: ISO 13485 for overall system quality and ISO 14971 for managing hazards. (The IEC drafting team proposes broadening a future edition's scope to health software, including non-medical software ([IEC design specification](#)).)

By enforcing a rigorous, process-oriented approach, IEC 62304 helps manufacturers avoid common software development pitfalls (such as inadequate testing, undocumented requirements, and uncontrolled changes) (quickbirdmedical.com). It also provides auditors with objective criteria: a completed IEC 62304-compliant life-cycle yields artifacts like software requirements specs, design reviews, and test reports, which collectively form a traceable design history. As one quality consultant observes, “a compliant QMS is the key to proving to auditors and regulators that your software was developed in a state of control from its initial concept all the way through to its final retirement” (www.qualityfwd.com).

FDA recognizes IEC 62304 Edition 1.1 for medical device software life cycle processes, within the scope stated in its [recognition record](#). EU presumption of conformity for a harmonised standard depends on publication of its reference in the Official Journal and applies only to the requirements it covers, as the [European Commission explains](#).

Safety Classification (IEC 62304:2006/2015)

A core concept in IEC 62304 is the **Software Safety Class**, which categorizes software by the potential severity of harm from its malfunction. There are three classes:

- **Class A:** The software cannot contribute to a situation that results in injury (or it does, but those injuries are considered acceptable and not even non-serious). In essence, Class A is the *lowest risk* category. These are features whose failure might cause inconvenience but not harm. An example might be a non-essential display feature or logging function. Even for Class A, the manufacturer must apply the standard's core processes, but less rigor is required in documentation and verification.
- **Class B:** Software where a failure could contribute to a *non-serious injury* or non-life-threatening harm. Think functions like basic alarms or record-keeping where an error might inconvenience the user or require minor medical attention but is unlikely to be life-threatening. Moderate rigor is mandated: more testing and documentation than Class A, but not the maximum.
- **Class C:** Software whose failure could lead directly to *serious injury or death*. These are the highest-risk functions – for example, control of a life-sustaining therapy (ventilator breath control, insulin dose calculation) or safety features (emergency shut-off). Class C demands the most stringent controls: comprehensive reviews, exhaustive testing, and extensive documentation.

IEC 62304's rules for determining the class are tied to risk analysis. In practice, one conducts a hazard analysis (often according to ISO 14971 methodology), then applies IEC 62304 criteria:

“When software failures cannot result in patient/user exposure to a hazard, or result only in acceptable residual risks, the software is Class A. When residual risks from a software failure become unacceptable, the software is Class B or C depending on whether the resulting injury is considered non-serious or serious, respectively.” (www.iss-ag.ch).

In other words (using the standard's own logic):

- If the failure contributes no hazardous condition or only an adequately controlled one ⇒ **Class A**.
- If the failure could contribute to an uncontrolled hazard leading to *non-serious* injury ⇒ **Class B**.
- If it could contribute to *serious* injury or death ⇒ **Class C**.

(IEC 62304's definitions of “serious” injury align with ISO 14971, and “harm” is interpreted in the broad sense used in risk management (www.nsf.org).) Table 1 summarizes these classes with illustrative examples.

IEC 62304 Class	Potential Harm from Software Failure	Example
A	No injury or only <i>acceptable</i> (minor) risk. Software failures <i>cannot</i> cause a hazard that leads to harm.	E.g. software modules for non-therapeutic monitoring, simple user interface functions.
B	Software failure <i>could</i> contribute to a hazardous situation resulting in non-serious injury .	E.g. an alarm function or intermediate data processing where error might cause transient or treatable harm.
C	Software failure <i>could</i> contribute to a hazardous situation resulting in serious injury or death .	E.g. control software for ventilator, critical drug dosage calculation.

“Non-serious” vs. “serious” per IEC is aligned with ISO 14971's definitions, generally meaning absence or presence of life-threatening consequences.

This risk-based classification drives the development process. For higher classes, IEC 62304 imposes increased documentation, design reviews, and verification activities. The standard's General Requirements section (Clause 4) explicitly ties the software's class to the rigor of subsequent processes. For instance, Class C software must produce a detailed Software Requirements Specification (SRS) and verified design for every software item, whereas for Class A less detailed artifacts may suffice (quickbirdmedical.com).

IEC 62304 also requires one to assume that *“software failure occurrence shall be assumed to be certain (probability = 100%)”* for the purpose of classification (www.iss-ag.ch). This means when analyzing risks, one treats any software bug as a definite event, focusing instead on how controls mitigate its impact. In short, the class determination is conservative: if any plausible software bug could lead to unacceptable harm, the higher class is assigned. This policy ensures rigorous treatment of uncertainty (justified given the complexity inherent in software) (www.iss-ag.ch).

FDA's June 2023 device-software premarket guidance uses Basic and Enhanced documentation levels, replacing the earlier Level of Concern terminology. These FDA documentation levels are separate from IEC 62304 safety classes ([FDA webinar transcript](#)).

Assigning the correct class is crucial. A wrong assignment (e.g. designating truly critical software as Class B instead of C) could leave insufficient controls in place, creating safety vulnerabilities (www.iss-ag.ch). Therefore, manufacturers often over-classify ("erring on the side of safety") to ensure compliance. Once the class is set, the title of IEC 62304 is effectively "*Software Class X life cycle processes*", where X is A, B, or C. The rest of the standard's requirements are interpreted through the lens of that class.

Key Processes of IEC 62304

IEC 62304 prescribes **five core process areas** for software life cycle and maintenance, each subdivided into specific activities. Table 2 outlines these processes with their fundamental objectives and key activities. The standard is structured so that Clauses 5–9 align with these processes:

- **Software Development Process (Clause 5):** Encompasses initial development from concept to release. This includes Software Development Planning (5.1), Requirements Analysis (5.2), Architecture Design (5.3), Detailed Design (5.4), Unit Implementation and Verification (5.5), Integration and Integration Testing (5.6), System Testing (5.7), and Release (5.8). Together, these form a waterfall-like lifecycle (which can be executed iteratively). Clause 5.5.5 software-unit verification applies to Classes B and C ([IEC drafting rationale](http://www.iss-ag.ch)). Software Release (5.8) mandates versioning and release notes.
- **Software Maintenance Process (Clause 6):** A controlled process for post-market software changes. When a device is in use, any change (bug fix, minor enhancement) must go through maintenance planning, re-analysis of risk, implementation, and testing. Select verification and validation activities through analysis of the change and its impact; urgency or a minor-fix label does not itself reduce applicable controls.
- **Software Risk Management Process (Clause 7):** This process integrates ISO 14971 risk management into software. It requires identifying potential hazards due to software failure, estimating their risk by combining hazard severity with probability, and specifying software safety requirements or controls to mitigate those risks. Notably, IEC 62304 calls for incorporating all risk-derived software requirements into the SRS (www.riskmanager.net). The standard also mandates that if external controls (e.g. hardware interlocks) exist, those are considered in the software's risk analysis, but only controls *outside* the failing software are allowed to reduce the assessed risk (www.iss-ag.ch).
- **Software Configuration Management (Clause 8):** Ensures integrity of the evolving software artifacts. It requires version control of source code, documentation, and builds. Every file (SRS, design diagrams, test scripts, source code, executables) must be under configuration control, with change procedures. This process ensures that any release can be traced back to a specific set of code and documents, which is vital for investigation of issues (e.g., in a recall, one can identify exactly which build was in the field).
- **Software Problem Resolution (Clause 9):** A formal defect and anomaly handling process. It requires logging any software aspect that does not meet requirements (from code bugs to documentation errors) as a "software problem report". Each item must be evaluated for severity, root cause, and appropriate action (repair, workaround). Known residual anomalies are evaluated for Classes B and C at release. At release, Clause 5.8.3 calls for evaluation of known residual anomalies for Classes B and C.

Each of these processes generates work products: plans, requirement documents, design records, test protocols, issue logs, etc. In aggregate, this documentation forms the Software *Design History File* (DHF), which regulators will scrutinize. A typical IEC 62304 development yields a tight trace: one can trace each software requirement (derived from risk analysis) through design, implementation, and testing. For example, risk-derived safety functions are codified as requirements in the SRS, implemented in code, and accompanied by dedicated verification tests. Attesting to authorities “we did that safety analysis according to 14971” without showing how those requirements entered the software is insufficient; hence IEC 62304 demands explicit traceability (www.riskmanager.net).

The standard does not *dictate* exactly *how* to implement processes. For example, Annex B of IEC 62304 (non-normative) explicitly states that agile development is compatible: plans may be iterative, and processes may recur as development progresses (www.riskmanager.net). One risk manager notes: “*planning is an iterative ACTIVITY that should be repeatedly reviewed and updated as development progresses*” (www.riskmanager.net). In practice, teams can use Scrum or other agile methods as long as they produce the required deliverables (e.g. backlog tied to hazards, sprint review of requirements). What matters is that nothing is skipped: every requirement must be verified, and every change must be controlled and justified.

Table 2 summarizes these processes, emphasizing the flow from planning to problem resolution. Each process may be implemented via any lifecycle model (V-model, agile iterations, etc.), but must be *defined* in the QMS. The table entries highlight the intention and example work products at each stage, with references to the standard's clauses and guidance sources.

Table 2: Overview of IEC 62304 Processes and Activities

Process (Clause)	Goal / Main Activities	Key Deliverables & Notes
5 – Software Development	Plan, design, implement and test new software. Activities include software development planning (scope, resources, interfaces) (www.riskmanager.net); requirements analysis (detail functional & safety requirements) (quickbirdmedical.com); software architecture and detailed design; coding (unit implementation); unit verification for Classes B and C under Clause 5.5.5; integration of units where applicable to the software safety class; system-level testing; and release preparation.	Software Development Plan; Software Requirements Spec (SRS) ; Architecture and Design docs; Unit test reports; Integration/System test reports; Release notes. Clause 5.5.5 software-unit verification applies to Classes B and C (IEC drafting rationale).
6 – Software Maintenance	Manage post-market changes (bug fixes, security patches, minor enhancements). Perform impact/risk analysis of requested changes, plan maintenance, implement, verify, and document solutions. A controlled process with activities selected for the maintenance change.	Updated versions of design docs and test results for the changes; maintenance plan; change control records. Must still re-evaluate risk and update SRS or risk analysis when needed.
7 – Software Risk Management	Identify and control risks due to software. Perform hazard analysis assuming software failure (probability=100%) and only external risk controls (www.iss-ag.ch). Derive software safety requirements to mitigate unacceptable risks, and integrate those into SRS. Verify that safety features/control measures meet intended function.	Risk Management File (hazard analysis, risk evaluation, risk control rationales) (www.iss-ag.ch); updated SRS with risk-derived requirements. Evidence that each software hazard's risk is acceptably controlled per ISO 14971.

Process (Clause)	Goal / Main Activities	Key Deliverables & Notes
8 – Configuration Management	Ensure integrity and traceability of software artifacts. Establish version control for source code, documentation, test scripts, executables, etc. Manage branching, baselines, and change control.	Configuration Management Plan; Version-controlled repository of all SW artifacts; change control records. (Compare every release to a baseline; document build environment.)
9 – Problem Resolution	Handle software anomalies and defects. Log every discovered bug or failure (in development or field). Assess severity, repeat/incidence, assign corrective action (fix, monitor, defer). Record the disposition of each problem. Update other processes if needed (e.g. risk analysis).	Software Problem Reports (bug reports) with analysis of cause and impact; Verification of fixes; Updated hazard/risk analysis if a new risk was identified. Document and evaluate known residual anomalies at release.

Table 2 Note: The applicable activities within these processes differ by class; Clause 5.6 integration and integration testing do not apply uniformly to Class A. For example, Class A might not require formal SRS annexes or design reviews to the same extent as Class C (www.qualityfwd.com). Class A activities follow the requirements applicable to that class; software unit verification under Clause 5.5.5 applies to Classes B and C.

General Requirements (Clause 4)

Before embarking on the specific processes, IEC 62304's Clause 4 imposes foundational QMS and risk management requirements. It requires **three general prerequisites**:

- 1. Quality Management System (QMS):** The manufacturer must have an established QMS for medical devices. While IEC 62304 does not detail the QMS itself, it notes that compliance with ISO 13485 is a way to meet this. In effect, IEC 62304 assumes that design controls (under ISO 13485:2016 as incorporated into FDA's current QMSR) are in place. The QMS should ensure that software processes are planned, executed, and documented consistently (quickbirdmedical.com) (www.qualityfwd.com).
- 2. Risk Management (ISO 14971):** A top-level risk management process per ISO 14971 must be in place. IEC 62304 does not itself derive risk, but it incorporates ISO 14971 by reference. The manufacturer must manage software hazards *through that system*. In practice, this means that hazards and risks from software are identified, evaluated, and controlled just like hardware risks (quickbirdmedical.com) (visuresolutions.com). Indeed, later clauses rely on ISO 14971 – for example, the risk controls and “severity of harm” come from that standard's definitions (www.nsf.org).
- 3. Software Safety Classification (Classes A/B/C):** As discussed above, each software item must be assigned a safety class (4.3). The class determines the rigor of documentation required (higher class ⇒ more documents, reviews, tests). The manufacturer has freedom in partitioning software into “items” and classes. IEC 62304 provides a decision tree for classification (Figure 1 in the standard); essentially, any uncertain case of unacceptable risk is escalated to the higher class.

By enforcing these prerequisites, IEC 62304 ensures that software is not developed in a vacuum. The broad quality framework (QMS) and risk processes are prerequisites, and software classification triggers the appropriate level of process control. In sum, Clause 4 declares “*the manufacturer must...*” do these things,

setting the stage for the detailed processes that follow.

Integration with Quality and Risk Standards

IEC 62304 does not stand alone; it operates within an ecosystem of standards. This integration is crucial for effective compliance and best practices.

ISO 13485 – Quality Management

ISO 13485 (Medical devices – Quality management systems) is the generic standard for device manufacturers' QMS. Clause 4.1 of IEC 62304 explicitly references ISO 13485: it requires the manufacturer to “operate a quality management system” (quickbirdmedical.com). In effect, all activities in IEC 62304 must be under a QMS umbrella. Notably, ISO 13485 Clause 7.3 (Design and development) overlaps substantially with IEC 62304's requirements: ISO 13485 demands design inputs/outputs, design review, verification, validation, and change controls, but doesn't specify the content for software. IEC 62304 can be viewed as *executing* ISO 13485's design controls specifically for software.

A QualityForward summary highlights this synergy: *“IEC 62304 focuses specifically on the software lifecycle, while ISO 13485 defines the requirements for the overall quality management system... The two standards are designed to work together in harmony”* (www.qualityfwd.com). In practice, companies often integrate IEC 62304's processes into their ISO 13485 QMS. For example, the IEC 62304 Software Development Plan becomes a subset of the QMS's design plan. Traceability matrices link ISO 13485 action items to IEC 62304 work products, and change control procedures cover both hardware and software changes. Indeed, one QMS consultant notes that building an “IEC 62304 QMS” is essentially part of meeting ISO 13485's design documentation and control requirements (www.qualityfwd.com).

Thus, compliance with ISO 13485 inherently supports IEC 62304: a strong QMS provides the mechanisms (document control, audits, CAPA, supplier controls, etc.) that software processes require. Conversely, IEC 62304 provides ISO 13485 with the specific guidance for *software* components that ISO lacks. For example, an ISO 13485 audit might check that a device's software design plan exists; an IEC 62304-focused audit will check that the plan includes software-specific risk analysis and verification plans.

ISO 14971 – Risk Management

ISO 14971 (Application of risk management to medical devices) is the international standard for device risk management. IEC 62304 integrates risk management in two ways:

- **Mandated linkage:** As noted, Clause 4.2 requires a risk management process per ISO 14971. This means all software hazards and associated harms must be assessed according to that standard's methodology (quickbirdmedical.com). In effect, ISO 14971 defines risk (probability × severity) and decides what is “acceptable”. IEC 62304 then says: *whatever ISO 14971's process finds, incorporate those results into software development.*
- **Embedded risk activities:** Clause 7 of IEC 62304 explicitly mirrors ISO 14971 phases but at the software level: identify software hazard causes (e.g. code defects), analyze severity (using ISO's harm categories), estimate probability (note IEC 62304 assumes 100% failure rate as worst-case, see

sidebar), and evaluate risk. Crucially, IEC 62304 requires that *risk controls* for hazards involving software be implemented and verified as part of the software itself (unless infeasible). For example, a hazard analysis may reveal that missing data validation could cause misdiagnosis; the control might be additional input checks coded into the software – which then becomes a formal requirement in the SRS (www.riskmanager.net).

IEC 62304 does **not** replace ISO 14971; rather it uses it as a companion. All hazard analyses should be performed under ISO 14971, and the results traced into IEC 62304 documentation. As one author emphasized, one of the “major confusions” is trying to separate IEC 62304 risk analysis from ISO 14971: in reality, “SOUP failure modes are in the same fault trees as your custom software so it would be easiest to analyze them simultaneously” (www.softwarecpr.com).

Other Standards

Several other standards interact with IEC 62304 in important ways:

- **IEC 60601-1 (Medical electrical equipment):** Older editions of IEC 60601-1 did not cover software, but IEC 60601-1-4 (treated as part of IEC 60601-1) provides requirements for programmable electronic safety-related systems. It emphasizes separate requirements for software development and verification. IEC 62304 complements this by giving much more detailed SW process requirements. In practice, compliance with IEC 60601-1's 3.4 (safety systems) implies implementing something like IEC 62304 for the software portions.
- **IEC 62366 (Usability):** This standard covers usability engineering of medical devices. Its application predominantly concerns UI software. IEC 62366 requires identifying use-related risks and designing the UI accordingly. While IEC 62304 focuses on the software **engineering process**, IEC 62366 focuses on human factors. They intersect: for instance, use-related hazard mitigations (from IEC 62366 process) may become software requirements (e.g. confirmation dialogs, error messages) that IEC 62304-related processes must implement and verify.
- **IEC 82304-1 (Health Software):** The standard covers design, development, validation, installation, maintenance, and disposal of health software products (IEC).
- **U.S. quality management regulation:** FDA's [Quality Management System Regulation](#) became effective February 2, 2026 and incorporates ISO 13485:2016 by reference. IEC 62304 addresses software life cycle processes; its use does not by itself establish QMSR compliance.

In summary, IEC 62304 sits at the nexus of software development, quality management, and risk management. It provides the *detailed execution* of design and verification controls for software that ISO 13485 and ISO 14971 define at a high level (www.qualityfwd.com). Adopting IEC 62304 means integrating these standards: e.g. the IEC 62304 Software Development plan becomes part of the ISO 13485 Design and Development plan; every risk identified in IEC 62304's Section 7 should tie back to the ISO 14971 risk file. This integration ensures consistency and avoids gaps or duplications. As one industry commentary states, using IEC 62304 is “*the clearest way to demonstrate to regulators that you have followed a rigorous and repeatable process*” (www.qualityfwd.com).

Regulatory Landscape and Global Perspectives

IEC 62304's impact and usage vary somewhat by region, but a common global theme is regulatory expectation of structured software processes. Below we outline how major regulators and regions treat medical device software and the role of IEC 62304.

United States (FDA)

In the United States, manufacturers must apply the requirements relevant to their device. FDA's [Quality Management System Regulation](#) became effective February 2, 2026 and incorporates ISO 13485:2016 by reference. FDA's [device-software premarket guidance](#) addresses submission documentation.

- **Recognized consensus standard:** FDA's [recognition record](#) lists IEC 62304 Edition 1.1 (2015) for medical device software development and maintenance. It excludes medical-device validation and final release from the standard's scope.
- **Device-software submissions:** FDA's June 2023 guidance recommends Basic or Enhanced documentation for device software functions in premarket submissions ([FDA guidance](#)).
- **Other FDA guidance:** The final [Computer Software Assurance guidance](#) concerns software used in medical-device production or the quality management system. It does not replace device-product software guidance.

European Union (CE Marking, MDR)

The EU Medical Device Regulation (MDR) governs qualifying medical devices, including software with a medical intended purpose. Manufacturers must assess applicable requirements for their devices and intended uses.

- **MDR and software:** Clinical evaluation is addressed in Article 61 and Annex XIV of the [MDR](#). A standard's use does not by itself establish compliance with all MDR requirements.
- **Classification:** MDR Annex VIII Rule 11 addresses software classification. IEC 62304 software safety classes A, B, and C serve a separate purpose; they are not MDR device classes.
- **Technical documentation:** Software life cycle records can support a manufacturer's technical documentation, subject to the requirements applicable to that device.
- **Harmonised standards:** The [European Commission](#) says voluntary use of a harmonised standard confers presumption of conformity only once its reference is published in the Official Journal, and only for the requirements it covers. Check the current published references before claiming a particular standard has that status.

The choice of standards should be assessed against the device's applicable MDR requirements and the current Official Journal references. IEC 62304 is a software life cycle process standard, not a general CE-marking authorization.

Other Regions and Global Harmonization

Beyond the US and EU, other regions also emphasize software standards:

- **International context:** Jurisdictions set their own medical-device requirements; manufacturers should check each market's current regulator requirements and standard-recognition records.

IEC 62304 can inform a consistent software life cycle process, while regulatory obligations remain specific to each jurisdiction and device.

Development Life Cycle Processes (Clause 5)

IEC 62304's *heart* is Clause 5, the Software Development Process. This clause spells out the required activities from inception to release. We summarize the main stages below (for more detail, see sources such as Malte Bucksch's guide (quickbirdmedical.com) and Visure's overview of risks (visuresolutions.com)).

5.1 Software Development Planning: The process begins with a formal plan. The Software Development Plan (SDP) must define the scope of work, responsibilities, interfaces (hardware, external systems, etc.), deliverables, schedules, and standards to be used. IEC 62304 allows iterative planning (the plan is an activity to be updated as development proceeds (www.riskmanager.net)) but each iteration must be documented. Plans also address configuration management, risk management integration, and problem-resolution processes.

Output: A documented SDP signed by authorized parties. (This plan often lives inside the higher-level Design and Development Plan of the QMS.)

5.2 Software Requirements Analysis: All system requirements (functional, performance, safety, regulatory) that pertain to software must be identified and documented in the SRS. This includes incorporation of risk control requirements from ISO 14971 analyses (for instance, if risk analysis says "system shall alarm on parameter X", it becomes a software requirement). The SRS should be clear, testable, and traceable. If multiple software components exist, there may be one SRS per component or subsystem.

Output: Software Requirements Specification (SRS) document, with traced links to higher-level system requirements and risk controls. Software requirements will later be verified via testing.

5.3 Software Architecture Design: With requirements in hand, a high-level architecture is created. This identifies software units, modules, and their interactions. Decisions about partitioning (for Class segregation or separate threads/processes) are made here. The architecture is documented (block diagrams, data flow descriptions) and reviewed. Crucially, safety-critical interactions should be identified: e.g. if a memory leak could cause system failure, the architecture should isolate or mitigate it.

Output: Architectural design documents, possibly UML diagrams or equivalent. Rationale for design choices (especially those affecting safety) is recorded.

5.4 Detailed Software Design: Each component from the architecture is then designed in detail. This stage includes algorithms, data structures, and interface definitions. Each software unit (often corresponding to a file or class in code) is designed to meet certain requirements. Documentation at this level might be in pseudo-

code or description of algorithmic logic, but it must be clear enough for implementation. Reviews should check consistency with architecture and requirements.

Output: Detailed design specifications for each software unit. The design must show how each requirement is addressed. Traceability from this design to the architecture/SRS is maintained.

5.5 Software Unit Implementation and Verification: Developers write the actual code for each unit defined in 5.4. For Classes B and C, Clause 5.5.5 addresses software-unit verification. Verification can be through code inspection/review and unit testing. Clause 5.5.5 software-unit verification applies to Classes B and C. For Class C especially, this might include rigorous code reviews and automated tests. Any defects found are logged and fixed through the Problem Resolution process (Clause 9).

Output: Source code for each unit, version-controlled in a repository. Unit Test Plans and Reports showing that code meets the detailed design. If code is auto-generated, the same verification activities apply to the generator or output.

5.6 Software Integration and Integration Testing: Next, the individual units are compiled/linked together into subsystems and eventually the full application. Integration may occur in stages (e.g. modules into libraries, libraries into executables). Clause 5.6 integration and integration testing requirements do not apply uniformly to Class A. Integration tests are designed to exercise interfaces and interactions, particularly those tied to safety features. Traceability between integrated system components and verified design elements must be maintained; if two safe units interact, the integration test must cover that interaction for safety.

Output: Integrated software builds. Integration Test Plans and Reports documenting testing of combined components against the design. Issues discovered feed into the Problem Resolution process.

5.7 System (Software System) Testing: Once all software units are integrated, full system-level testing occurs. The system is tested against the SRS: this includes functional tests, performance tests, stress tests, and any user-acceptance testing that may be required. If the software interacts with hardware, tests may involve the actual device or a hardware simulator. For SaMD, tests may simulate clinical scenarios. Verification here demonstrates that *all* software requirements (including those from risk analyses) are met. High-risk features need particularly exhaustive testing, including failure modes.

Output: System Test Reports showing pass/fail for each requirement-derived test. A traceability matrix tying tests back to requirements is often used to show coverage.

5.8 Software Release: Finally, the released version of the software is formally documented. This includes generating a version identifier, software release notes listing changes, user manuals, and packaging deliverables. For stand-alone software, the actual executable or installation package is prepared; for embedded software, the releases are often binary/firmware images. The configuration is baselined for archives. All development artifacts for that release (code, docs, test results) are formally frozen in configuration control.

Output: Released software package and User Documentation; Finalized versions of SRS, design, test, and risk files. A Release certificate or similar document may be used to formally sign off the release.

Clause 5.5.5 software-unit verification applies to Classes B and C. One concise summary states the required core activities (development planning, requirements analysis, architecture design, detailed design, unit implementation, integration & integration testing, system testing) in a checklist manner (quickbirdmedical.com). At all times, traceability is emphasized: as the RiskManager article notes, *“If requirements are placed on the software in the risk management process, these must be evident such that the risk control measures are also traceable”* (www.riskmanager.net). In practice, this means using traceability matrices or tools that link hazards → requirements → design → tests. Modern eQMS or ALM tools often support this end-to-end traceability with links across documents (for example, a requirement item in the SRS document links to code commits and test cases). Auditors typically review a traceability matrix as a quick check of completeness.

IEC 62304 does not specify a development model (it even explicitly states no particular “model” is required (www.riskmanager.net)). Whatever approach a team takes, the applicable Clause 5.6 activities depend on software safety class. For example, Agile teams might do mini-cycles of requirements, design, coding, and testing (a “sprint”) with iterative planning updates as described in Clause 5.1 (www.riskmanager.net). After each sprint, they could produce incremental SRS updates and test reports. As long as the formal outputs exist and are reviewed, the standard is satisfied. Indeed, Annex B of IEC 62304 encourages tailoring processes to the project, as long as all safety objectives are met (www.riskmanager.net).

Support Processes (Clauses 6–9)

IEC 62304 allocates Clauses 6–9 to cross-cutting processes that support or run parallel to development:

Clause 6 – Software Maintenance Process: Once a device is on the market, software bugs or requested enhancements will inevitably arise. Maintenance planning selects appropriate development activities for each type of maintenance. It requires planning the maintenance (including coordinating with post-market surveillance), re-evaluating risks of any changes, implementing the fix, and re-verifying. A key point is that maintenance for Class B/C requires a process similar to development (with hazard review and testing) even for bug fixes. Rapid fixes are allowed, but they must still be tested and documented. However, companies should note that auditors will scrutinize maintenance records just as much: if a patch introduces a new feature, it may trigger full IEC 62304 compliance for that change.

Clause 7 – Risk Management Process: This clause refers back to ISO 14971 but focuses on software. It instructs the manufacturer to follow “the risk management process in IEC 14971” for hazards due to software failures. Components include identifying software-initiated hazard sequences, assessing risk, and implementing software risk controls (like fail-safes). Clause 7 provisions include tracking risk controls to software elements (e.g. encoding a risk control as a software safety requirement), and updating risk analysis after verification results if needed. It also requires documenting residual risk acceptability. Essentially, Clause 7 ensures risk management is not just a system-level checkbox but a software-integral activity.

Clause 8 – Configuration Management Process: Software configuration management (CM) goes beyond what most hardware dev processes require. IEC 62304’s Clause 8 demands formal CM on all software artifacts: a structured source control system, baselining and version numbering, access control, build and release procedures. This aligns with best practices in software engineering (e.g., “build pipeline”). The goal is that any released software version can be reproduced and traced to its source. Clause 8 typically references that all documents (requirements, code, tests, design) should be under CM. For example, older versions of

62304 highlight a “configuration baseline” of requirements and code prior to implementation. The QualityForward blog emphasizes that an electronic QMS (eQMS) or integrated ALM tool is nearly indispensable for handling this complexity (www.qualityfwd.com).

Clause 9 – Problem Resolution Process: When defects are found (either in development or in the field), Clause 9 provides the defect correction lifecycle. Problem reports require evaluation and documented disposition, including release evaluation of any known residual anomalies for Classes B and C. Notably, this process must include documenting any anomalies found in verification (not just in production) and evaluating known residual anomalies for Classes B and C at release. For field anomalies, there should be a linkage to the risk analysis: if a software bug is found in use, the organization must analyze whether it affects any software safety function. The maintenance record (Clause 6) and problem resolution both generate traceable artifacts: every bug fix is a change that needs updating in SRS/tests.

Together, these processes ensure software integrity throughout its operational life. For example, the combination of Configuration Management and Problem Resolution means there is no “grey market” of untracked patches – any change to code is managed, documented, and approved. And Software Risk Management is ongoing: it’s not just a one-off initial analysis, but is revisited whenever a new problem is discovered or a change is proposed.

Implementation and Best Practices

Implementing IEC 62304 can be challenging in practice, especially for organizations newer to medical devices or smaller companies. We discuss key considerations and recommended practices:

- **Quality Management System Infrastructure:** Successful IEC 62304 compliance hinges on a solid QMS foundation (often ISO 13485 certified). Quality manuals and SOPs should explicitly cover software: e.g. separate document control processes for code, defined roles for software architects, guidelines for software reviews. Many companies find it helpful to adopt or adapt electronic Quality Management (eQMS) or Application Lifecycle Management (ALM) tools that provide built-in traceability. As one consultant recommends, an eQMS can *“enforce your development workflows, manage links between requirements, code, tests, and risks, and provide a complete, auditable design history file”* (www.qualityfwd.com).
- **Training and Competence:** Staff must be trained on IEC 62304 requirements. This includes developers (to produce required docs and tests), risk managers (to integrate ISO 14971 into software context), and auditors. Cross-training is also valuable: engineers should understand regulatory context, and QA should grasp basic coding. Organizations often send key personnel to IEC 62304 courses or workshops. The QualityForward blog notes that 62304 is not just for engineers; *“your entire quality and regulatory organization must understand and support”* its requirements (www.qualityfwd.com).
- **Process Tailoring:** IEC 62304 allows tailoring. For example, agile teams can document each sprint’s results (backlog items tied to code, tests, etc.). RiskManager publishes guidance explicitly showing how agile fits: planning is iterative, and tasks can be done concurrently as long as reviews and traceability

are maintained (www.riskmanager.net) (www.riskmanager.net). It is wise to avoid process “gaps” that could anger auditors – e.g., even if beta testing is done informally, formal system verification against requirements should still be documented.

- **Software Quality Tools:** Automated testing, static code analysis, and continuous integration can help satisfy verification needs. For example, unit test frameworks and coverage tools can provide objective evidence that all code paths (especially safety-critical ones) have been exercised. FDA provides guidance to help manufacturers design and maintain cybersecure products ([FDA](#)). Use of recognized coding standards (MISRA C, CERT guidelines) is encouraged for higher classes as it simplifies defect analysis.
- **Managing SOUP and Outsourced Code:** IEC 62304 addresses SOUP function and performance requirements in Clause 5.3.3. Best practice is to treat SOUP in combination with custom code: analyze SOUP failure modes in the same risk analysis. One SoftwareCPR blog advice is “*analyze its failure modes and design in risk controls*,” rather than isolating SOUP documentation as a separate yard sale (www.softwarecpr.com) (www.softwarecpr.com). If using open-source or commercial off-the-shelf components, conduct a clear quality assessment (supplier evaluation, code provenance, known issues tracking) and include that in the risk file. Even if design history for SOUP is unavailable, manufacturers must still ensure it meets safety needs through validation testing and additional safeguards.
- **Legacy Software:** Many companies have “grandfathered” software developed before IEC 62304’s publication. Edition 1.1 includes requirements for showing conformity of legacy software to IEC 62304. The IEC design specification proposes moving legacy software provisions to an informative annex and changing mandatory language to guidance ([IEC design specification](#)). Practically, firms should inventory legacy systems, assess them under ISO 14971, and create retrofit verification traces showing that even historical functions have been sufficiently tested. This is often a weak point during audits if overlooked.
- **Clinical Input and Verification:** For software with clinical implications, IEC 62304 requires that the interface between clinical requirements and software specs be traced. Collaborating with clinicians during requirements and testing (e.g. used in user acceptance tests or clinical simulations) is prudent. For example, an ECG analysis algorithm should be checked on representative patient data. Any discrepancies or edge-case failures should be captured and risk-reviewed. This clinical perspective often reveals practical hazards that an engineer might miss.
- **Documentation Rigor:** One of the challenges teams face is avoiding “documentation bloat” while meeting the standard. Strategies include using templates (e.g. preformatted SRS, risk files), clear revision histories, and combining related documents (when allowed). However, cutting corners is risky: many audit findings come from missing traces or undocumented verification steps. A helpful mindset is that *every output except the final code itself is “evidence” for auditors*. For instance, instead of describing a test in narrative form only, provide actual test protocols and reports with pass/fail and revision numbers.

Validation Boundary

IEC 62304 does not cover validation and final release of the medical device, even when the medical device consists entirely of software ([FDA recognition record](#)).

Process Evidence and Analysis

This section focuses on the types of evidence a manufacturer should assemble to evaluate software life cycle processes.

- **Evidence to review:** Requirements traceability, risk-control verification, configuration history, problem reports, and maintenance records can reveal process gaps. No quantitative trend is asserted here without an identifiable primary dataset.

Discussion: Implications and Future Directions

IEC 62304 has become the backbone for safe medical software, but the landscape is evolving. In this section, we explore implications for stakeholders and emerging trends:

Implications for Stakeholders

- **Manufacturers:** Establish software planning, risk management, verification, configuration control, and maintenance processes that fit the device and the applicable regulatory requirements.
- **Engineers:** Software teams should document requirements, design decisions, verification, and changes so that work can be reviewed and traced.
- **Regulators:** FDA's final [Computer Software Assurance guidance](#) is scoped to computers and automated data processing systems used in medical-device production or quality management systems.
- **Healthcare providers and patients:** Clear requirements, verification records, and documented changes help manufacturers evaluate whether software continues to perform as intended.

Future of the Standard

The IEC design specification provides guidance for drafting a second edition of IEC 62304. The following changes are proposals, subject to the final standard:

- **Scope Expansion:** The IEC drafting team proposes a standard covering health software, including non-medical software and software that is a device or in a device ([IEC design specification](#)).
- **Proposed Rigor Levels:** The IEC design specification maps current Class A to Level I and Classes B and C to Level II in its proposal ([IEC design specification](#)). Manufacturers should reassess their processes when a final edition and applicable regulator transition rules are available.
- **AI/ML Considerations:** The design specification identifies AI as a possible topic for Annex C's discussion of related standards ([IEC design specification](#)).

- **Cybersecurity Integration:** FDA's cybersecurity guidance addresses section 524B of the FD&C Act for cyber devices ([FDA guidance](#)).
- **Legacy and migration:** Manufacturers should assess existing software and later changes against applicable requirements and the standard edition they choose to use. A future standard revision does not itself set a regulator transition period.

Future Implications

The evolution of IEC 62304 has broad implications for the future of medical device software:

- **Global Convergence:** The IEC drafting team proposes expanding the standard to health software, including non-medical software. A unified standard could encourage best practices across the board. For manufacturers crossing between regulated and unregulated markets, this enables smoother process integration.
- **New Competencies:** Developers will need to incorporate competence in AI/ML risk. Traditional software engineers will have to work more closely with data scientists, statisticians, and clinicians. Risk analyses will involve probabilistic errors from models, not just code bugs.
- **Scope of FDA CSA:** FDA's [Computer Software Assurance guidance](#) addresses computers and automated data processing systems used in medical-device production or the quality management system.
- **Vendor Lock-In of Practices:** The widespread adoption of IEC 62304 means that contractors or suppliers lacking medical experience may struggle to adapt. Conversely, companies with embedded instances of IEC 62304 in their culture will enjoy smoother partnerships. A long-term effect may be that new employees in medtech are expected to already know IEC 62304 fundamentals – akin to expecting civ. engineers to know building codes.
- **Healthcare Outcomes:** Ultimately, the goal of IEC 62304 is safer devices. Studies on software-induced patient harm will be a key metric for its effectiveness. If properly implemented, one would expect a downward trend in software-related recalls and adverse events over time. Monitoring could include analyzing FDA's Medical Device Reporting (MDR) data for software error trends. At present, explicit trends are hard to isolate, but future safety studies may cite IEC 62304 compliance as a factor in reducing incidence of software errors in devices.

Conclusion

IEC 62304 describes a medical-device software life cycle framework, including development, maintenance, risk management, configuration management, and problem resolution. FDA recognizes Edition 1.1 (2015) for the scope in its [recognition record](#).

Its A/B/C safety classes guide the extent of software life cycle activities. Manufacturers should address quality management and device risk management alongside the standard's software processes.

Future revisions and emerging technology may change how teams implement those processes. Check the applicable edition, regulatory requirements, and current guidance for each device and market.

References

- Malte Bucksch, “IEC 62304: Software life cycle processes for medical devices”, QuickBird Medical (Sep. 2020) (quickbirdmedical.com) (quickbirdmedical.com).
- Daniel Mannion, “IEC 62304 Edition 2 – Big changes in SaMD requirements”, Intuition Labs/8foldGovernance (Mar. 10, 2025) (8foldgovernance.com) (8foldgovernance.com).
- “IEC 62304, Draft of the 2nd Edition”, NSF Life Science News (Feb. 7, 2025) (www.nsf.org).
- FDA, *Medical Device Software as a Medical Device (SaMD)*, FDA.gov. (Defines SaMD, IMDRF role) (www.fda.gov) (www.fda.gov).
- Visure Solutions, “IEC-62304 Risk Management For Medical Devices” (blog) (visuresolutions.com) (visuresolutions.com).
- European Commission, *Harmonised Standards (MDR)*, EC Health (accessed 2026) (health.ec.europa.eu) (health.ec.europa.eu).
- QualityForward, “IEC 62304 QMS Checklist for Medical Software Teams” (Oct. 26, 2025) (www.qualityfwd.com) (www.qualityfwd.com).
- SoftwareCPR, “SOUP – Can’t Live Without It!” (Aug. 2021) (www.softwarecpr.com) (www.softwarecpr.com).
- Blue Goat Cyber, “IEC 62304 and Medical Device Cybersecurity” (Dec. 27, 2025) (bluegoatcyber.com) (bluegoatcyber.com).
- Visure Solutions, *IEC 62304 Risk Management (Part 1)* (blog) (visuresolutions.com) (visuresolutions.com).
- IntuitionLabs, “IEC 62304: Medical Device Software Life Cycle Processes” (intuitionlabs.ai) (AI-based source, cited sparingly).
- FDA Fusion Center, “Recognized Consensus Standards: Medical Devices” (Jan. 2019) (www.accessdata.fda.gov) (for IEC 62304 recognition).
- IMDRF, “SaMD: Clinical Evaluation – Guideline” (2016) and associated documents (risk framework, etc.) (www.fda.gov).
- European MDCG, “Guidance on software labeling & classification” (MDCG 2019-11 Rev.1, 2025) (punktum.net).
- ISO 13485:2016, “Medical devices – QMS – Requirements” (referenced by IEC 62304) (www.qualityfwd.com).
- ISO 14971:2019, “Medical devices – Risk management” (referenced by IEC 62304) (quickbirdmedical.com).

Source links

Links cited in this article, in order of first appearance. Full addresses are provided for readers using extracted text.

1. <https://intuitionlabs.ai/articles/fda-samd-classification-ai-machine-learning>
2. https://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfStandards/detail.cfm?standard__identification_no=38829

3. <https://intuitionlabs.ai/articles/ai-medical-device-compliance-eu-mdr-ai-act>
4. <https://www.softwarecpr.com/2021/08/soup-cant-live-without-it/>
5. <https://www.riskmanager.net/en/2022/08/10/agile-iec-62304/>
6. <https://assets.iec.ch/public/sc62a/IEC%2062304%20Ed.%202.0%20Design%20Specification.pdf>
7. <https://quickbirdmedical.com/en/iec-62304-medical-device-software/>
8. <https://www.qualityfwd.com/blog/iec-62304-qms/>
9. <https://www.iss-ag.ch/en/blog-articles/software-failure-probability-key-to-iec-62304-safety-classification>
10. <https://visuresolutions.com/blog/medical-devices/iec-62304/>
11. https://health.ec.europa.eu/medical-devices-topics-interest/harmonised-standards_en
12. <https://bluegoatcyber.com/blog/iec-62304-and-medical-device-cybersecurity/>
13. <https://www.fda.gov/medical-devices/digital-health-center-excellence/software-medical-device-samd>
14. <https://webstore.iec.ch/en/publication/22794>
15. <https://www.nsf.org/ie/en/life-science-news/iec-62304-draft-of-the-2nd-edition-optimized-safety-classes-extended-scope-for-healthcare-software-and-ai>
16. <https://www.fda.gov/media/170714/download>
17. https://assets.iec.ch/public/sc62a/N0166_IEC62304_ED2_ChangeRationales_CoverLetter.pdf
18. <https://webstore.iec.ch/en/publication/26120>
19. <https://www.fda.gov/medical-devices/postmarket-requirements-devices/quality-management-system-regulation-qmsr>
20. <https://www.fda.gov/regulatory-information/search-fda-guidance-documents/content-premarket-submissions-device-software-functions>
21. <https://www.fda.gov/regulatory-information/search-fda-guidance-documents/computer-software-assurance-production-and-quality-management-system-software>
22. <https://eur-lex.europa.eu/eli/reg/2017/745/oj/eng>
23. <https://www.fda.gov/consumers/consumer-updates/medical-device-cybersecurity-what-you-need-know>
24. <https://www.fda.gov/regulatory-information/search-fda-guidance-documents/cybersecurity-medical-devices-quality-management-system-considerations-and-content-premarket>
25. <https://8foldgovernance.com/iec-62304-edition-2-big-changes-in-samd-requirements/>
26. https://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfStandards/results.cfm?ascapilotyn=off&category=&effectivedatefrom=&effectivedateto=&organization=&pagenum=100&productcode=&recognitionnumber=&referencenumber=®ulationnumber=&sortcolumn=pad&start_search=201&supportingdocsyn=off&title=
27. <https://punktum.net/insights/software-as-a-medical-device-samd-explained-mdr-rule-11-iso-standards/>

THE PUBLISHER

About IntuitionLabs

IntuitionLabs is an AI consulting, custom software development and data engineering firm serving pharmaceutical, biotechnology, medical-device and other life-science organizations. We work with clinical, regulatory, medical-affairs, commercial, quality and IT teams to connect technology decisions with the work people need to accomplish.

AI consulting and adoption

Our AI enablement services cover readiness assessments, use-case selection, governance and policies, team workshops, adoption measurement and ongoing advisory support. We help organizations structure the information layer behind AI: source material, context, permissions and maintained knowledge that make generated answers useful and reviewable. Private LLM inference and hosted AI options support teams evaluating how to operate AI with appropriate control over their data and infrastructure.

Software, data and life-science workflows

IntuitionLabs develops custom software for pharma and biotech, integrates enterprise systems, and builds data engineering and business intelligence solutions. Areas of focus include AI agents, regulatory research, medical writing, medical affairs, CMC information, competitive intelligence and clinical-document workflows. Our eTMF intelligence work includes cross-system reconciliation and inspection-readiness support.

Enterprise platforms and regulated delivery

We provide Veeva services, application support, managed services, integrations and custom applications, alongside enterprise content work involving platforms such as Egnyte. For regulated workflows, our services include GxP enablement, computer-system validation and software development addressing 21 CFR Part 11 requirements. The applicable controls, validation responsibilities and acceptance criteria are defined for each engagement.

Work with IntuitionLabs

Explore [AI enablement](#), [pharma and biotech software development](#), [data engineering and BI](#), and [Veeva services](#). [Contact IntuitionLabs](#) to discuss your workflow, information sources and implementation needs.

IntuitionLabs publishes educational research to help life-science teams make informed technology decisions. Coverage of a product or organization does not imply a client relationship, endorsement or partnership.

Website: <https://intuitionlabs.ai>

This article is educational. Verify consequential medical, legal, financial, regulatory and technical decisions with appropriate professionals and the cited primary sources. Company services are described separately from the article's research findings.