



How to Develop Compliant Medical Device Software

Controlling the Software Development Life Cycle for Medical Devices

October 2019

Abstract / About the Author:

Over the past two decades, perhaps the most significant change to the medical device industry has been the incorporation of software into a burgeoning number of medical devices. While this development trend has resulted in increased functionality and sophistication, including better control of devices, more power to end-users, and phenomenal gains in diagnostic and usage data gathering and dissemination, it has also raised a host of issues and questions.

One of Velentium's Principal System Architect & Engineers, Satyajit Ketkar (Sat), will explain how companies can adopt these best practices, as well as show in detailed steps the way Velentium accomplishes these specific tasks internally from the setup, inputs, outputs, testing, wrap-up, and handoff. Sat has nineteen (19) years of engineering experience with seven (7) of those years within medical device design. A majority of his career has revolved around electrical, firmware, software and systems engineering but recently he spent over eighteen (18) months working for a European Union notified body. This experience allowed him to see product development in a different way, teaching him how to review and audit products for safety and, quality, performance, and security.

Contents

- Is Software a Device? 3
- Who is Responsible? 3
- Setup 4
- Configuration Management..... 4
- User Needs 5
- Risk Analysis 6
- Vulnerability Assessment..... 6
- Requirements..... 6
- Architecture and Detailed Design 7
- Implementation 7
- Code Reviews, Static Analysis, & Unit Testing 7
- Integration Testing..... 8
- Micro Penetration Testing 8
- Human Factors Study 8
- System Testing 9
- Documentation & Traceability..... 9
- Development Wrap-Up..... 9
- Maintenance Planning & Future Development 10

Introduction

While it's not simple to understand how software safety, functionality, and control work in the sphere of medical devices, the FDA and ISO have provided significant guidance, which can become the basis of a development process that meets regulatory and user requirements. By establishing requirements, developing a process that fits them, following that process carefully, and steadily producing artifacts which document each step along the way, medical device manufacturers can ensure that the software component(s) of their devices will perform to expectations without causing costly delays or roadblocks to development, approval, release, or post-market.

In this series, Sat will provide a high-level overview of the controls needed to develop medical device software that meets accepted standards and merits regulatory approval.

Is Software a Device?

The FDA's [definition of a medical device is clear](#). Since 2010, [the FDA has been equally clear](#) that software than in any way interacts with a medical device or works alone in the attempt to diagnose, cure, mitigate, treat, or prevent a disease, is itself a device.

Once the software was classified as a device, it, in turn, became necessary for software to [comply with 21 CFR Part 820](#) (even the sections that do not specifically mention software). Any medical device that contains or is composed of software is not compliant with regulatory requirements unless the

software has undergone risk management, configuration management, requirements management, design controls, verification/validation, and other requirements of Part 820.

Each of these components of software development must occur within a quality management system and include the required documentation to prove compliance.

Who is Responsible?

When the scope of compliance requirements surrounding medical device software is understood, it becomes evident that software development has a high potential for risk / benefit impact on patients and end-users. No wonder, then, that software development controls can make or break device approval! Devices that are otherwise Part 820 compliant and have useful clinical data on safety and efficacy will still be denied market approval if the software development process and documentation are not compliant.

Medical device companies often outsource software development for strategic or financial reasons. However, it is still the device manufacturer of record that is responsible for showing that the software in their device meets regulatory requirements. The device manufacturer that outsources software development, therefore, has two choices: either lead the software consultant through a development process that will



SOFTWARE DEVELOPMENT
CONTROLS ARE CRUCIAL!

ensure compliance or hire a consultant that is knowledgeable of requirements and has developed its own certified process by which to ensure compliance.

In the medical device industry, it is simply not enough to hire competent programmers for device software development. Skilled programmers are necessary, but they must also work within a disciplined framework of a software development process tailored to the FDA and other regulations.

Setup

Acceptable software development follows a series of repeatable steps that ensure that all requirements are met. There is no single “correct” software development process, but any good process must incorporate the requirements of 21 CFR Part 820, IEC 62304, the CDRH guidance on software validation, FDA cGMP, ISO 14971, and all FDA guidances that elaborate on Part 820. The process needs to be both disciplined and flexible in order to accommodate both the FDA’s [Software Levels of Concern](#) rating system and IEC 62304’s [Software Safety Classification](#) system.

Here’s a sample process consisting of ten components, only one of which includes actual coding:

- Configuration Management
- User Needs (End User Assessment)
- Risk Analysis & Security Analysis (Vulnerability Assessment)
- Requirements
- Architecture & Detailed Design
- Implementation (Coding)
- Code Reviews, Static Analysis & Unit Testing

- Integration Testing
- Micro Penetration Testing
- Human Factors Study
- System Testing
- Documentation & Traceability

When regulatory requirements are fully understood, and each component of software development is carefully implemented, the resulting medical device software meets standards of functionality, safety, and regulatory compliance.

Configuration Management


First things first: before you even begin looking at project specifics, you’ve got to make some decisions (and document them!) regarding configuration management for the project.

Configuration management systems ensure that development proceeds in a controlled, consistent, traceable, auditable manner. It includes change requests, defects tracking, and release management. During medical device development, it is essential that released packages are tightly managed and reproducible for the full lifetime of the device.

Version control during development comprises a significant element of configuration management. Version control facilitates parallel development by keeping your team working from the same central source of truth, making it easier to reconcile conflicts, and helping to protect the master project from any bugs or mistakes. However, version control alone isn't sufficient to meet coding standards for medical device software. Configuration management goes further, determining version control workflows, tracking tasks and issues and documenting *in medias res* decisions and review findings, defining and enforcing release states to prevent "configuration drift," and so on.

Given that, it should go without saying that configuration management applies to more than just code and software resources. It also includes all of the process records and quality artifacts produced, including design, risk, and requirements documents.

Selecting the right combination of CM tools and defining the CM process for your project is non-trivial. Fortunately, there are abundant



VELENTIUM CAN HELP WITH
DEFINING YOUR CM
PROCESS!

resources available, including the contact form on our website. As consultants who've contributed to a

wide variety of medical device projects for well-established industry majors as well as small start-ups, aimed at a numerous indications, we'd be happy to offer tips, discuss best practices and lessons learned, and relate what we've seen work well for the smoothest development experience with the fewest speedbumps in the approval process.

User Needs

An evaluation of user needs begins with asking extensive questions about who the end-user of the device is, what they are capable of, what their limitations are, and how their interaction with software design might affect device functionality or safety.

It is essential to consider whether a given device could have multiple audiences for different use cases. For example, the end-user of an implantable device is the patient, but audience consideration must include the physician, any potential caregivers who will be interacting with the device, the scientists who will conduct studies on the device, and so on.

By carefully considering end-user and audience at the outset of software development, the overall course for the software is set. This is also a crucial initial component in [human factors studies](#), which should be considered throughout the design process.

Unless your device consists exclusively of software, it may well be that user needs have been investigated and documented at the system level, and perhaps even broken down from there to the software level, before the development team is brought into the project. In that case, it's vital that the lead developer and systems engineer carefully review the use cases and risk profile together, asking whether any software-specific user needs have been missed or haven't been clearly described. This review will build directly into the Requirements phase of the development lifecycle.

Risk Analysis

Risk management is not a single phase; instead, it begins as Requirements are defined and runs parallel to the development process for the remainder of the project. Risk Analysis and Management requires a multi-functional team of experts to determine how the software will affect risk, how software could mitigate other identified risks in the device as a whole and hazards created by the software itself. Outcomes of risk management must be documented, and each design change must result in concurrent risk management review. (See [ISO 14971 for guidance on risk management](#)).

Two major artifacts that you'll want to begin working on concurrent with Design activities, which we'll cover in the next post, are a Preliminary Hazard Analysis (PHA) and a Failure Modes & Effects Analysis (FMEA). Exploring these two documents in-depth could become a whole blog series in itself, but at a high level, generating these for your software involves critically examining the software design from both a top-down (PHA) and a bottom-up (FMEA) perspective. The PHA begins with a list of potential hazards the user could experience, classifies those hazards, and traces each risk to a design requirement or development activity that mitigates it. The FMEA begins with a list of software components, determines how each could fail, describes the effects of each possible failure and determines its hazard classification, and traces each impact of each failure mode to a requirement or activity that mitigates it. The difference is subtle but essential for demonstrating acceptably thorough risk management.

Vulnerability Assessment

As the FDA and ISO define it, "risk" refers to the device's potential to harm the patient. One form of risk entails the device or device elements functioning as intended, failing to function as expected, or being misunderstood and/or misused, and that's what the PHA and DFMEA risk analyses focus on. Another form of risk is the device's potential to cause harm due to interference from malicious cyber activity.



Over the past two years, the FDA, ISO, and other medical device regulators have increasingly [clarified their expectations](#) concerning secure development, as it applies both to systems and software. Foundational for fulfilling most of these requirements is a [Vulnerability Assessment](#), which looks critically at your intended design to identify all of the known ways a system with that design could be compromised or rendered unavailable through malicious cyber activity. The output of that assessment may then be used to define requirements and refine software design to ensure these vulnerabilities are mitigated to an acceptable level.

Requirements

Software requirements derive directly from User Needs and are scoped and informed by Risk and Security Analyses. Essentially, this

stage of the process is about knowing the right questions to ask in order to arrive at the correct set of requirements. Requirements gathering involves collating several avenues of research, including functionality, safety, usability, and regulations.

Outputs of this phase build into the Design History File (DHF), which is required as part of the device approval submission. These outputs might all be labeled Requirements, or they can be broken down by area – for example, you [may need to differentiate](#) between requirements derived from user needs versus those required to meet applicable standards.

Architecture and Detailed Design

There are many different architectural approaches for designing a software system. Examples include data-driven, event-based, rules-based, state-based, service-oriented, and more. Parsing out which to use for a given project is beyond the scope of this series; from a controlled-development perspective, what's most important is that your architecture and detailed design documents be directly traceable back up to the requirements and risk documents you produced during the input phases.

To this end, it's helpful to design a numbering system that can maintain continuity between these documents. If your numbering convention includes a 1:1 component, it creates high visibility from designed modules back to requirements, enabling rapid confirmation that every requirement is covered by appropriate software elements,

and every software element has been analyzed for risk and appropriately mitigated.

Implementation

At this point in the software development process, actual coding begins. The coding platform(s) your developers will use was determined as a part of the requirements and risk management phases, and functional units were identified in detailed design. Thanks to your robust configuration management system, different functional units can be worked on in parallel by different teams or individuals at this point.

It is important to note that there should be regular risk review meetings during this phase in order to keep risk management forefront throughout.

Code Reviews, Static Analysis, & Unit Testing

Each part of the code should be tested as it is written. It is critical – especially in large, complex systems – to make sure it is possible to exercise individual units as thoroughly as possible. This may require creating a simulated system that can interact with the code or even a physical test apparatus. Test needs will be determined both by software design and risk management and will have been previously documented as a subsection of the system Testing Plan (not covered in this series).

Unit testing is a formal discipline. There should be documentation that shows the planned test procedure, date tested, personnel who performed the test, and

UNIT TESTS SHOULD BE
TRACEABLE BACK TO DESIGN
AND REQUIREMENTS DOCS!

results of the test. Unit tests should be traceable back to design and requirements documents to prove complete

coverage of the requirements with tested software. In addition, there should be documented proof that unit test personnel were adequately trained and capable of conducting unit testing.

To learn more about Code Reviews and software Testing, refer to [our blog series on Static and Dynamic Analysis](#). We also have a free white paper available to download, as well as [configuration instructions and a plug-in kit](#) for our preferred Unit Testing tool, Parasoft.

Integration Testing

As individual units pass testing, move to test them as integrated units. You've proven that these units function as expected in isolation; now, you are systematically integrating and testing them to verify that they work as expected in combination with one another. Throughout this process, your Test Plan will require that you repeatedly ask "what could possibly go wrong" with each unit integration and ensures that you have devised reliable means to check.

Micro Penetration Testing

Penetration testing systematically attacks your system under controlled conditions mimicking anticipated use cases and environments to identify cybersecurity vulnerabilities. Traditionally, this is performed at the end of the development lifecycle, just prior to or concurrent with verification and validation testing. However, this practice produces a report too late in the development lifecycle to fix many vulnerabilities cost-effectively. Although it does provide the information needed for FDA and intentional regulations, it does not do it in a timely fashion optimized for most software development companies' business models.

Valentium has successfully pioneered an alternative approach, dubbed "[micro penetration testing](#)," which scopes cybersecurity test activities to particular areas of the overall system and performs them concurrent with implementation. With this approach, reports generated can be fed back immediately into designing and implementing cybersecurity mitigations before they become expensive to address.

Human Factors Study

It is essential that [human factors engineering](#) be taking place throughout the software development process. For example, if the device includes a user interface, show sample screens to potential users as soon as they are produced. It is much easier to identify human factor issues early in the process.

Human factors testing includes both qualitative and quantitative testing. In qualitative testing, users are presented with the whole process and must use the device without being guided in each step. What errors in use are noted? What seemed to confuse or frustrate the user? Was there anything that distracted them?

Quantitative testing implements a disciplined workflow where the users are asked about every detail of the user-interface—every screen, button, control, readout, etc. Prior to this testing, as part of risk management, a predetermined percentage of positive results that are required on any unit or set of units must be identified and documented. Any aspect of human factors testing that does not meet that percentage must be reworked and taken back through the process.

System Testing

Now it is time to test everything as a complete system. The system test procedure should be planned ahead of time (as part of the overall Test Plan, at the requirements and design phases). There should be clear delineation, based on your previously-determined requirements documents, of what a “complete” software system looks like.

System testing should also include a formal code review by knowledgeable programmers who have not been a part of the development of the code under scrutiny. The discipline of presenting code to another programmer often identifies weaknesses in architecture that the author could not see in the midst of coding.

Documentation & Traceability

As should be clear by now, even though we’re presenting it here as if it is a final step, rigorous documentation following accepted industry standards must occur throughout the development process. Though many developers dislike documentation, in the eyes of a regulator, if something is not documented, it did not happen. Controlled software development requires not only that the software has safety and functionality, but that the medical device’s Design History File (DHF) contains complete records of each safety and functional element’s design, implementation, and testing. Furthermore, DHF artifacts must include traceability so that reviewers, regulators, and auditors can follow the granular development of individual elements from each process phase to the next.

Development Wrap-Up

As we have seen, a reliable software design process for medical devices is disciplined and methodical. While there are significant creativity and coding skill that goes into the best software medical devices, it must occur within a framework that understands that safety and efficacy are paramount to all other concerns.

As a final thought, you will note that the software development process presented here is relatively restrictive. Changes cannot be made to any component without affecting all of the others—a change to a system requirement requires a new risk analysis, an

updated to the detailed design documents, new testing of the changed unit as well as the system as a whole, etc. While such changes can be made, careful planning at the outset by an experienced team can minimize such changes and subsequent delays.

Maintenance Planning & Future Development

Developing well-controlled software for medical devices doesn't end with market approval. Once a system is designed and planned, it must be version-locked as part of the implementation. Planned releases need to be gated against the documented design, and any ideas, feedback, or additional development work need to be isolated into a "Phase II" repository for future release. This disciplined approach ensures that the software portion of a medical device will not hold back completion or approval.

While it is possible that responsibility for the wellbeing of your software in the field may be

transferred from a development team to a maintenance team, even emergency updates, like critical bug fixes and vulnerability patches, must be planned for and deployed in a controlled manner. Even though you can't necessarily anticipate the content of these updates, you can and must leverage the know-how from your development team and your maintenance team to define the process by which the need for updates will be determined, classified according to response type and time, securely developed & delivered, and verified for efficacy after deployment.

In other words, initial release can't be the final goal of your software design, with activities taking place afterward being treated as an afterthought or a task for someone else. In order to truly mitigate risks and vulnerabilities of your design, controlled development must include planning for controlled maintenance.