Congenius

Congenius Whitepaper

An introduction to Design Control for Software

July 2024

By Dr Dirk Hüber

Edited by Hannah Matthews

Contents

1. Introduction

Introduction

The number of regulations, standards, and guidance documents specific to software in or as a medical device (SiMD, SaMD) is substantial and growing. The recent rise in regulatory documents that have been developed and set into force coincides with the sharp increase in SiMD and SaMD development within the medical devices industry.

Over the past few years, many companies that previously developed purely mechanical or unconnected devices with electronics and embedded software (eSW) have started to develop connected devices, apps, and/or other digital solutions. Venturing into this new territory comes with its challenges – with even non-medical apps or digital solutions being regulated under the same principles as medical device software (e.g., in terms of cybersecurity and data protection).

Furthermore, whilst developing the source code for a certain functionality (e.g., in an app) is a reasonable effort, compliantly documenting and testing the source code takes a multiple of this duration, not to mention the time and budget required for ongoing software maintenance. When the effort is underestimated, costs explode, and deadlines are missed.

In this whitepaper we outline some basics regarding design control for software that will facilitate an understanding of why the regulatory requirements for medical device software are in force, and why they make sense. Within the following pages we explore:

- The different **processes to develop and test software** (e.g., the classic waterfall vs the V-model)
- **Cybersecurity and data protection** requirements for connected devices and software
- The importance of **software updates & configuration management**
- The regulatory requirements for software taken from external sources, such as off-the-shelf **(OTS)** software and software of unknown provenance **(SOUP)**, including open-source software **(OSS)** and free open-source software **(FOSS),** and the need for ongoing **maintenance** of these kinds of software
- The aspects to consider when **purchasing software** – even if it's free of charge

A few notes before we dive in...

Before we start, here are some useful definitions to help with your understanding of this paper:

Software as a Medical Device (SaMD)
Software intended to be used for one or more medical purposes that perform these purposes without being part of a hardware medical device.

Software in a medical device (SiMD)
Software which is integrated into a medical device to control its performance or provide specific functions.

=

eSW (Embedded Software)
Software which is integrated into a medical device to control its performance or provide specific functions.

Unconnected device
A device with electronics and eSW that in normal and intended operation does not connect to any other device or to any network, but functions autonomously. An unconnected device has no interface to other devices or systems (neither wired nor wireless) that is accessible to any user during normal and intended operation.

Note: Often a device may have an interface to other devices or systems not accessible for the normal users or patients, but only accessible for maintenance and service personnel. Such devices are connected devices in the regulatory sense, which should not be overlooked.

2. Software Development Process

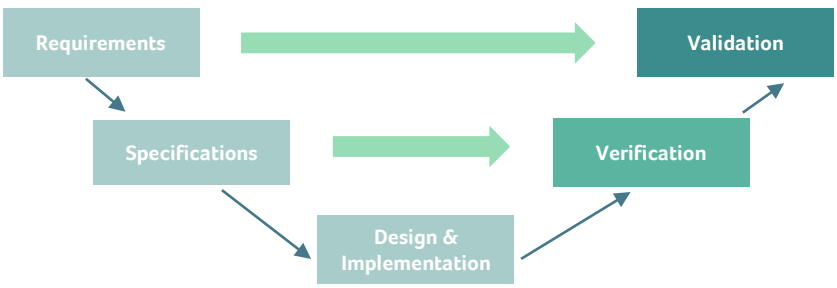
Software Development Process

The Classic Waterfall or V-Model

In the **classic waterfall model**, both the product and the manufacturing process are basically **developed in one go**, i.e., without loops (at least in an ideal world):

Step 1 | All requirements for the product are collected
Step 2 | All requirements are specified
Step 3 | The design for the product is then derived from the specifications
Step 4 | Verification tests are undertaken, to prove that the design meets the specifications
Step 5 | Validation tests are undertaken, to prove that the product meets the requirements

Similarly, for the manufacturing process, first all process requirements are collected, then the equipment and the process are specified, and from the specifications the equipment and process design is derived. Next, the equipment and process are qualified against their specifications, and finally the process is validated against the requirements. This is depicted in the following (simplified) figure:



Thus, each step is ideally carried out in one go. Such an approach is appropriate if the implementation of the design is very costly, i.e. if tools and equipment for manufacturing are expensive - as is usually the case for physical devices. In such cases it's important to ensure you have the correct design for each part, component, and the device itself before committing to any investment into costly tools and equipment, so that the probability to pass verification and validation is very high. Any change to tools and equipment would create significant additional costs.

The disadvantage of such an approach is its rigidity. Any change while underway is expensive in time and effort – the later in the process, the more so. Even if investments have not yet been made, updates of development records might become costly. The change of a single specification may necessitate the update of numerous other documents if development is already far progressed.

However, the benefit of protecting the investment far outreaches such disadvantage for parts and components that require high investments like, for example, plastic moulded parts.

Software Development Process

Agile Models

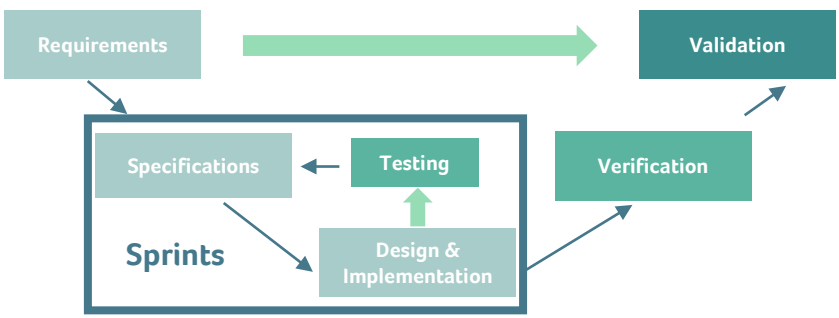
In contrast to the Classic Waterfall or V-model, **Agile Models are built for flexibility**.

For software, there is no investment to be made for tools and equipment that is not already needed for development itself. Thus, there is no investment that needs to be protected from late changes. However, what does prove costly in software development is testing the source code only when development is finished. This would lead to several findings in testing, which can prove difficult to fix, given that any individual fix may cause another issue elsewhere.

Software is a highly complex affair, developed by many software engineers in parallel, and frequently derived from external sources – often only as executables with unavailable source code. Consequently, software can only be controlled with a very structured approach.

One part of the solution is to structure the software into software items and software units, which are developed independently from each other and then integrated into the next higher level and finally into the whole software system.

The other part of the solution is to develop software stepwise, whereby the steps are called sprints. In each sprint, a predefined set of specifications is implemented and then tested. Findings of tests performed in each sprint are either fixed immediately or put into an issue list and fixed in a later sprint. There are various agile models, but in principle they look as shown below:



Software Development Process

Agile Models (continued)

The crucial part in software development is the testing. Various test methods are employed for software, for example:

- **Code reviews |** Each piece of code is reviewed by another software engineer before it is checked in for further testing.
- **Static code testing |** Automated check of the source code against coding guidelines. Coding guidelines ensure maintainability, but also certain security aspects.
- **Unit testing |** Automated tests of the functionality of a software unit. Unit tests are scripted by the software engineers and simulate interfaces to other software units. Often, not 100% of the source code can be reached by unit tests. In such cases, code that cannot be reached has to be checked by other methods, e.g., by code reviews.
- **Integration tests |** Software units are integrated into a software package and their functionality tested together with other software units in a test bed. External interfaces to the software package are simulated.
- **System tests |** The software as a whole is tested in its real run time environment, interfacing with other systems outside of the software.
- **Penetration tests |** Security testing, automated and manual, to detect potential security leaks.

Some of these tests are performed within each sprint, so that no untested software is handed over to the next sprint. Issues found during testing might be fixed in a later sprint, especially if finding the root cause and/or fixing the issue will take time.

Integration, system, and penetration tests are executed if and when the software has reached a respective state.

One consequence of such an agile development model is that the development documentation is also developed during the sprints. For example, rather than developing the full set of specifications and completing the design as in the waterfall model, specifications and design may be developed within each sprint for the scope of that sprint.

This also has consequences for other processes like risk management which to some extent should also follow the sprints.

It is important to understand that regardless of how rigorous the testing is done in each sprint and later during verification and validation, “bug-free” software does not exist – a point we will come back to later in this paper.

Software Development Process

Devices with Embedded Software

The above also applies to embedded software. However, embedded software is developed in parallel to the physical device into which the software is embedded. The physical device, comprised of the electronics and the mechanical part, is usually developed following the waterfall model.

Thus, the challenge is to synchronise the development activities between these different constituents of an electronic device. In this synchronisation task it's important to not only consider the different approaches in the development process, but also the different time frames needed to develop the constituents. The synchronisation points must also consider the interdependencies in the functionality of the constituents (especially between the embedded software and the electronics on which it runs), e.g., when planning system tests.

Software Development Process

The Toolchain

Design control for software is not only different from design control for classic medical devices due to the different development approach, but also since software is developed in a so-called toolchain.

The toolchain consists of all the tools necessary to develop and test the software, e.g., a tool for requirement engineering, issue handling, source code repository, software development kits for integration of OTS and SOUP, compiler, builder, run time environments, static code analyser, unit test tool. Essentially, to develop software, you need many tools that are themselves software.

Often, tools used in the tool chain also offer support for other, not directly software-related processes, like risk management, complaint handling, and change management. The toolchain itself is usually comprised not only of OTS, but also of SOUP.

To ensure that the software in the toolchain functions as intended, Computer System Validation (CSV) or Computer System Assurance (CSA) apply. The challenge is to find pragmatic approaches for CSV and CSA that sufficiently balance effort with value.

Attempts to fully validate a toolchain element are often futile – for example, it is simply not possible to validate a compiler, even more so, if you have not developed it yourself. Most toolchain elements should be treated as infrastructure.

That said, like any software, the toolchain is never free of bugs. Therefore, like the product software itself, the software of the toolchain must also be monitored and maintained, as will be described in the following chapters.

It’s important to note, that for the toolchain, cybersecurity and data protection must also be addressed.

3. Cybersecurity & Data Protection

Cybersecurity & Data Protection

A medical device containing software can only be registered if cybersecurity and data protection have been adequately addressed.

However, cybersecurity and data protection are not unique to medical device software - they apply to all software. Thus, medical device companies should not only consider these topics for their product related software, but also for example, their IT infrastructure (as indeed any company should do according to ISO 27001).

Cybersecurity

Established methods exist for how to identify vulnerabilities and threats. That said, an experienced expert in cybersecurity is needed to apply such methods in a cybersecurity assessment together with the software development team, to identify potential vulnerabilities and threats, assess their criticality, and define appropriate mitigations.

For each identified vulnerability and threat, it is important to check whether it may lead to a harm to a patient, user, or the environment. If so, the associated risk must be evaluated in the risk analysis. Thus, for a medical device, cybersecurity assessment and risk analysis are connected.

Data Protection

All data that are created, received, stored, or processed by a medical device must be classified, and measures must be taken to adhere to data protection laws in the markets into which the medical device shall be marketed. Note that data classification and laws are different in different markets (e.g., GDPR for the EU and HIPAA for US).

4. Software Updates & Configuration Management

Software Updates & Configuration Management

When a classical medical device has been successfully registered, manufacturers usually hesitate to change it, since a change may become costly. Changes to tools or production equipment require investment; they need to become qualified and the manufacturing process re-validated. Product verification or even validation in the form of usability or clinical studies might have to be performed, authorities may need to be informed, and a re-submission might be required. All this can become time consuming and expensive. Therefore, manufacturers tend to make as few changes on classic medical devices as possible, to keep costs down.

For software, costs can only be minimised if the software is constantly updated. The reason being, that even the most rigorous testing will never discover all deficiencies in a software; rather, many deficiencies are discovered only when the software is used (see also our explanation in Chapter 6 - Software Maintenance). These deficiencies must be addressed and fixed via updates. Since all software items are constantly updated, these updates may create issues with other software items and new deficiencies may consequently be introduced. As such, bug fixing is a constant task.

Similarly, the environment of the software is constantly changing, and so the software must be adapted accordingly. Not only does the hardware and the operating system on which the software runs continually update, systems to which the software is connected also change constantly. What's more, the methods to attack software are also continuously improving, causing new threats.

If these changes around the software are not constantly addressed, the software will soon cease to function properly and become open to attack. In other words, the medical device software may no longer be safe, secure, or able to fulfil its intended purpose. If this happens, not only will the manufacturer's reputation be jeopardised, but product recall will also likely be required. Both occurrences will prove very costly to the manufacturer.

Software Updates & Configuration Management

Software Updates

Software is updated for one of the following reasons:

1. **New or improved functionality**
2. **Bug fixes**
3. **Removal or mitigation of vulnerabilities**
4. **Improving maintainability**

Only software updates due to the first reason are at the discretion of the legal manufacturer. Updates necessary for the other reasons are mandatory – and the FDA is very clear about this. Since FDA requires manufacturers to perform such updates, and to do so in a timely manner (because updates due to reasons two and three may be critical to functionality or security), FDA does not require that such updates are reported. For more on this, see the FDA guidance "Deciding when to submit a 510(k) for a Software Change to an existing device".

Manufacturers must be able to perform such updates quickly – if the criticality is high, within days, to prevent further damage. This requires a fast change process and the ability to build, test, release, and deploy an update on short notice and within a short time period. The organisation and its processes need to be prepared for this.

Software Updates & Configuration Management

Configuration Management

In configuration management, you not only need to document the software releases, but also be aware of which releases are in the market. If a new software release is deployed into the market, not all previously deployed software is always updated to the new release. This may be the case if updates cannot be done remotely, or if remote updates cannot be enforced, e.g., for apps.

The number of combinations of software releases in the market may increase especially quickly if a manufacturer has several software in the market that interact with each other (e.g., eSW in a connected device that interacts with an app and a cloud). For software that runs on various hardware and operating systems not under control of the manufacturer (e.g., for apps running on various smartphones with various versions of the operating system), configuration management and ensuring proper functionality might be challenging.

As such, when planning a new release of a software component, compatibility of the new release with existing releases (and their components) and existing runtime environments in the market must be checked. In case the new release is not fully backwards compatible, measures must be taken to ensure that the update works for all releases in the market.

For that reason, it’s worth using a configuration management tool rather than, for example, Excel sheets, to ensure optimum efficiency.

5. OTS & SOUP

OTS & SOUP

Nowadays, an application consists of only a small amount of self-developed source code. Most of the source code comes from OTS software and SOUP, including OSS. The task of the software engineer is to correctly integrate such software into the application so that it performs as intended for the application.

The same applies to the software in the toolchain: the toolchain consists mainly of OTS software and SOUP, including OSS. Thus, in many ways, the toolchain must be treated just as the medical device software itself.

The above means that software mainly consists of source code that is not under the control of the manufacturer of the software application. Nevertheless, the manufacturer is fully accountable for the medical device software they place onto the market, including its OTS and SOUP components.

There are regulatory requirements and guidelines available on how to handle OTS and SOUP in general and in medical device software in particular. In the following two chapters, we will focus on two situations where software must be handled differently to physical devices:

- In maintaining the software and,
- To ensure maintainability, in purchasing of software

These aspects apply to all software, be it self-developed, OTS, or SOUP, and be it software in the product or in the toolchain. However, for OTS and SOUP special considerations and actions must be taken. We will explore this further on the following pages.

6. Software Maintenance

Software maintenance

Software is never “finished”. In contrast to a physical device, it undergoes constant change and therefore must be maintained. This is because, as mentioned in Chapter 4, software always has bugs and vulnerabilities. There are two reasons for this:

Firstly, it is not possible to test software in all states that might occur when the software is operated, regardless of how diligently the testing is done. There will always be unforeseen possibilities for things to go wrong. And fixing a bug often enough creates a new bug. Thus, **bug fixing is an ongoing task that only stops if the software is removed from the market**.

This applies to both self-developed software components, and OTS/SOUP components. These components are either developed by commercial companies (OTS), or by open (OSS) or more or less closed communities (other SOUPs). In any case, these components are usually not intended for use in medical devices or for the development of medical device software and therefore do not adhere to medical device regulations.

Secondly, the means to attack a software constantly improve. Thus, if a software’s risk to vulnerability is judged acceptable today, it might become unacceptable tomorrow. **Improving a software’s cybersecurity is an ongoing task that only stops if the software is removed from the market.**

Software maintenance

To mitigate the risks associated with using such software components (in general and in medical devices), medical device software manufacturers may profit from the actions described below:

Commercial manufacturers of OTS regularly publish information on bugs, vulnerabilities, or at least on updates. This information has to be monitored and evaluated. The schedule for such activity should be weekly (during development, when the software is not yet released to the market, a monthly schedule might be sufficient). Manufacturers of medical devices should also publish information on bugs, vulnerabilities, or at least on updates for their own medical device software, and actively inform users about bugs, vulnerabilities, and updates at least if required by regulations.

For the large commercial manufacturers of software, organisations have been established that audit the manufacturer on a regular basis (e.g., annually or biannually) and have access to privileged information about the software. To access this information the medical device manufacturer must become a member of such an organisation. To be granted full access to this information, the manufacturer must often be a member for a certain time and/or be actively engaged.

Organisations have also been established that collect information about bugs and vulnerabilities as well as regarding available updates for OTS and SOUP, which they publish in their databases. These databases are searchable with dedicated software, that may be configured and parameterised such that only relevant information is provided. To access this information, the medical device manufacturer must become a member of such an organisation. The search of the databases for bugs, vulnerabilities, and updates should be done weekly (during development a monthly schedule may be sufficient), and the results evaluated.

SOUPS are often maintained by a community. Manufacturers should become a member of the community and be actively engaged, to access the available information about the SOUP.

Adequate resources and processes to perform these software maintenance activities must be planned for and provided by management.

7. Purchasing of Software

Purchasing of software

To facilitate the performance of software maintenance as described in the previous chapter, certain aspects must be considered when purchasing software. Thereby “purchasing” also takes place if the software is free of charge, i.e. FOSS.

To ensure, as described below, that purchased software can be adequately maintained, downloading software should be blocked for all employees including software engineers. Software should be downloaded only by a small, dedicated, group of employees (e.g., in IT) after formal approval of the purchase.

To approve the purchase of a software, the following aspects (among others) should be checked:

- ✓ Is the software well established and acknowledged as industry standard for the intended usage?
- ✓ How stable and secure is the software?
- ✓ Do organisations and communities as described in the previous chapter exist?
- ✓ Are these organisations and communities active?
- ✓ Is the manufacturer already a member of these organisations and communities?
- ✓ Or, if not, is it worthwhile to become a member of these organisations and communities, considering the effort the membership will require?
- ✓ Have the licencing conditions been checked? For example, is the manufacturer obliged to make its own software freely available when using FOSS?
- ✓ Are there any ethical concerns connected with the software?

Once approved, these aspects should be reviewed on a regular basis, e.g., annually, if there is any change in their status. If the criteria to use a certain software are no longer fulfilled, a replacement should be initiated.

Adequate resources and processes to perform these purchasing activities must be planned for and provided by management.

8. Conclusion

Conclusion

To bring medical device software onto the market and to maintain it in the market differs significantly from medical devices that do not contain software or are not connected.

To ensure timings and budget are met, decision makers need to be aware of such differences. Existing processes must be adapted, new processes for software should be implemented within the QMS, engineers, quality staff, and personnel from other functions with different skill sets and experience need to be employed, and the organisation must adapt accordingly. Several processes and functions we have not mentioned in this paper are also affected, e.g., human factors, clinical, post market surveillance, regulatory, and IT.

An app or any other digital solution that's regulated as a medical device cannot be developed and deployed without careful planning, and a comprehensive understanding of the requirements for software products – including the consequences of such requirements.

Our eHealth team at Congenius can help to facilitate the successful development of your medical device software, offering support regarding SaMD and SiMD for physical devices, eHealth, mHealth and Digital Health. Simply get in touch to start the conversation.

Congenius

Should you have a design control for software related challenge, feel free to get in touch with our eHealth team.

Further technical detail is also included in our whitepaper **How to develop SaMD**.

Congenius AG
Riedstrasse 1
CH - 8953 Dietikon

E: info@congenius.ch
T: +41 44 741 04 04

congenius.ch

Revision 1 – July 2024